DAT410 Module 2 Assignment 2 – Group 26

Yahui Wu (MPMOB) (15 hrs)
yahuiw@chalmers.se
Personal number: 000617-3918
Tianshuo Xiao (MPMOB) (15 hrs)
tianshuo@chalmers.se
Personal number: 000922-7950

January 31, 2023

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions

Reading and reflection

Yahui Wu

Netflix, as a world famous company, stated an open competition called Netflix Prize in 2006. The competition was aimed to find the best collaborative filtering algorithm which could reduce the root mean squared error (RMSE) by 10% of the test data which contained more than 100 million customer generated movie ratings. The characteristics of the difficult questions posed by Netflix could be summarized in three points. First, there was a large size of data which means building a module to handle the data efficiently is challenging. Secondly, nearly all the potential user-item pairs had no rating. Machine learning methods designed for complete data situations must be modified or abandoned. Thirdly, the pattern of observed data was very nonrandom. Besides, the factors affected a particular user's rating. Based on the characteristics of this challenge. a variety of models are needed to be built and improved. To achieve the goal, the authors of $Lessoons from the Netflix Prize$ mainly focused on the most common and typical collaborative filtering algorithm, one is k-NN and the other one is latent factor models.

The two main tools for collaborative filtering addressed different levels of structure in the data. Neighborhood methods could be effective when exploring the localized relationships. There are two approach to achieve neighborhood method. The item approach is to predict ratings based on ratings of neighboring items from the same user. The user approach predicts ratings based on ratings of the same item by similar users. Latent factor models based on matrix factorization factorizes the $m \times n$ rating matrix R which is the product of the user factor $P_{m\times f}$ and $Q_{n\times f}$. Both two typical collaborative filtering methods have their own advantages and disadvantages. Latent factor methods perform well when estimating overall structure and in this case, it relates simultaneously to most or all movies. However, it shows limit while exploring the relations among closely related set of movies. Neighborhood methods are easy to achieve without many parameters to tune and they provide a concise and intuitive justification fro predictions. However, the standard neighborhood methods are restricted by some coefficients selecting. In the paper, the author concluded three aspects on improving existing models. First, deepening known methods. Second, combining multi-scale views of the data. Third, combining explicit rating information with implicit rating behavior. Standard neighborhood methods predict the rating r_{ui} for movie i by user u by weighted average approach which could raise some concerns. First, the similarity function s_{ij} is arbitrary. Second, standard neighborhood methods do not account for interactions among neighbors. Third, the interpolation weights may cause overfitting. Last, variability differing substantially may hinder methods working. To overcome the difficulties, the authors computed interpolation weights w_{ij} derived directly from the ratings residuals (r_{ui}-b_{ui}).

The authors improved latent factor methods by applying a richer Gaussian prior. Meanwhile, they integrated the local viewpoint to reduce the limit caused by the characteristics of the previous latent factor methods. Alternatively, the authors integrated the binary viewpoint to improve prediction accuracy by taking NSVD method.

Tianshuo Xiao

Netflix has a recommendation system, Cinematch, which can suggest items of interest and enjoyment to people based on their preferences. The recommendation system is v critical to both the subscribers and Netflix because it can provide movies they are interested in and the company can make a profit by offering a service to subscribers.

The Cinematch recommendation system automatically analyses cumulative movie ratings each week, using Pearson correlations with all other movies to determine a list of "similar" movies that can predict how much a movie will be liked. When judging the accuracy of Cinematch, the root mean square error (RMSE) of the system's predictions is calculated and compared to the actual ratings provided by the

user.
When selecting the training set, they used two separate random sampling processes to first make up the entire Prize dataset, followed by a subset of quizzes, tests and probes used to evaluate the performance of the participants' systems. It was formed by randomly selecting a subset of all users who provided at least 20 ratings between October 1998 and December 2005. The qualifying set is formed by selecting the most recent set of scores for each user randomly selected from the full Prize dataset. These scores are randomly assigned, with equal probability, into three subsets: quiz, test, and probe. But no team had reached 10% improvement level after one year.
Among the models with larger promotion, they chose models that could compensate for their respective drawbacks, including nearest neighbor models and latent factor models. We found that best neighborhood models typically used for signal prediction which often ignore the vast majority of rating by users of interest.
In latent factor models, it aims to discover and interpret the potential features of the observed ratings. It performs an SVD decomposition of the matrix R for matrices with no deficient values to obtain bast rank-f. Then defined two rank-f matrices P_{m*f} and Q_{n*f}. The matrix R obtains the f most significant features and ignores the noise. At the same time, they voted based on user opinion, for which they called "the binary view of the data", a good complement to the above model.
For the weights for Neighborhood Models, the main objective of the model is to predict unobserved user ratings for films. In a set of neighboring items N(i; u) and all the items in it must have been rated by u (User u, item i, weight $r_u i$). However, this method can cause problems such as independence from other variables and overfitting, so the article proposes a method to optimize the weighting method.
In the article, it proposed an alternative way of weighting, by using a general weighting instead of a weighted average. By this way when a neighbor lacks information, it can fade the information of its neighbors and therefore needs to perform the calculation of the interpolation weights (least squares). This method has been shown experimentally to improve accuracy.
For Latent Factor Models, this model is mainly used to find connections between users and films. It is generally used minimize the regularized cost function. In order to avoid overfitting of the regularization parameters, the article proposes to use alternating least squares scheme. This approach can be used to improve accuracy by increasing the foundation of the model. In addition, when there are not enough ratings available, we can avoid over-fitting by reducing the factors to baseline values by means of a priori values. In the article, they used NSVD to integrate the binary viewpoint instead of the local viewpoint. Then, the parameters are estimated by minimizing the gradient descent of the associated regularized squared error. The models created in this way will be more flexible and accurate.

Implementation

```
import numpy as np
import pandas as pd
user_review = pd.read_csv("./movie_reviews/user_reviews.csv")
df_t = user_review.transpose()
df_drop = user_review.drop(user_review.columns[[0,1]],axis=1)

movie_name = list(user_review.columns[2:])
user_review.columns = list(range(-2,2000))

rating_list = []
name_list=[]
for i in range(0,600):
    name_list.append([user_review.iloc[i,1]]*2000)

name_list = sum(name_list,[])


movie_list= (user_review.columns[2:].values.tolist()*600)

rating_list=sum(df_drop.values.tolist(),[])
new_data = list(zip(name_list,movie_list,rating_list))

df_new = pd.DataFrame(new_data, columns = ['Name','Movie','Rating'])
df_new = df_new[~df_new['Rating'].isin([0])]
df_new.to_csv('new_data.csv')

import sys
import random
import math
import os
from operator import itemgetter

from collections import defaultdict

random.seed(0)


class ItemBasedCF(object):
    ''' ItemCF recommender system to recommend 5 movies for specific users '''

    def __init__(self):
        self.trainset = {}
        self.testset = {}

        self.n_sim_movie = 20
        self.n_rec_movie = 5

        self.movie_sim_mat = {}
        self.movie_popular = {}
        self.movie_count = 0

        print('Similar movie number = %d' % self.n_sim_movie, file=sys.stderr)
        print('Recommended movie number = %d' %
              self.n_rec_movie, file=sys.stderr)

    @staticmethod
    def loadfile(filename):
        ''' load a file, return a generator. '''
        fp = open(filename, 'r')
        for i, line in enumerate(fp):
            yield line.strip('\r\n')
```

```
            if i % 100000 == 0:
                print ('loading %s(%s)' % (filename, i), file=sys.stderr)
        fp.close()
        print ('load %s succ' % filename, file=sys.stderr)

    def generate_dataset(self, filename, pivot=0.7):
        ''' load rating data and split it to training set and test set '''
        trainset_len = 0
        testset_len = 0

        for line in self.loadfile(filename):
            list_rating = line.split(',')
            # split the data by pivot
            user, movie, rating = list_rating[1], list_rating[2], list_rating[3]

            if random.random() < pivot:
                self.trainset.setdefault(user, {})
                self.trainset[user][movie] = rating
                trainset_len += 1
            else:
                self.testset.setdefault(user, {})
                self.testset[user][movie] = rating
                testset_len += 1

        print ('split training set and test set succ', file=sys.stderr)
        print ('train set = %s' % trainset_len, file=sys.stderr)
        print ('test set = %s' % testset_len, file=sys.stderr)

    def calc_movie_sim(self):
        ''' calculate movie similarity matrix '''
        print('counting movies number and popularity...', file=sys.stderr)

        for user, movies in self.trainset.items():
            for movie in movies:
                # count item popularity
                if movie not in self.movie_popular:
                    self.movie_popular[movie] = 0
                self.movie_popular[movie] += 1

        print('count movies number and popularity succ', file=sys.stderr)

        # save the total number of movies
        self.movie_count = len(self.movie_popular)
        print('total movie number = %d' % self.movie_count, file=sys.stderr)

        # count co-rated users between items
        itemsim_mat = self.movie_sim_mat
        print('building co-rated users matrix...', file=sys.stderr)

        for user, movies in self.trainset.items():
            for m1 in movies:
                itemsim_mat.setdefault(m1, defaultdict(int))
                for m2 in movies:
                    if m1 == m2:
                        continue
                    itemsim_mat[m1][m2] += 1

        print('build co-rated users matrix succ', file=sys.stderr)

        # calculate similarity matrix
        print('calculating movie similarity matrix...', file=sys.stderr)
        simfactor_count = 0
        PRINT_STEP = 2000000
```

```
        for m1, related_movies in itemsim_mat.items():
            for m2, count in related_movies.items():
                itemsim_mat[m1][m2] = count / math.sqrt(
                    self.movie_popular[m1] * self.movie_popular[m2])
                simfactor_count += 1
                if simfactor_count % PRINT_STEP == 0:
                    print('calculating movie similarity factor(%d)' %
                          simfactor_count, file=sys.stderr)

        print('calculate movie similarity matrix(similarity factor) succ',
              file=sys.stderr)
        print('Total similarity factor number = %d' %
              simfactor_count, file=sys.stderr)

    def recommend(self, user):
        ''' Find K similar movies and recommend N movies. '''
        K = self.n_sim_movie
        N = self.n_rec_movie
        rank = {}
        watched_movies = self.trainset[user]

        for movie, rating in watched_movies.items():
            for related_movie, similarity_factor in sorted(self.movie_sim_mat[movie].
    items(),
                                                           key=itemgetter(1), reverse=
    True)[:K]:
                if related_movie in watched_movies:
                    continue
                rank.setdefault(related_movie, 0)
                rank[related_movie] += similarity_factor * int(float(rating))
        # return the N best movies
        return sorted(rank.items(), key=itemgetter(1), reverse=True)[:N]

    def evaluate(self):
        ''' print evaluation result: precision, recall, coverage and popularity '''
        print('Evaluation start...', file=sys.stderr)

        N = self.n_rec_movie
        #  varables for precision and recall
        hit = 0
        rec_count = 0
        test_count = 0
        # varables for coverage
        all_rec_movies = set()
        # varables for popularity
        popular_sum = 0

        for i, user in enumerate(self.trainset):
            if i % 500 == 0:
                print ('recommended for %d users' % i, file=sys.stderr)
            test_movies = self.testset.get(user, {})
            rec_movies = self.recommend(user)
            rec_movies_num = [i[0] for i in rec_movies]
            if user in ['Vincent', 'Edgar', 'Addilyn', 'Marlee', 'Javier']:
                movie_list=[]
                for i in range(len(rec_movies_num)):
                    movie_list.append(movie_name[int(float(rec_movies_num[i]))])
                print(user)
                print( movie_list)
            for movie, _ in rec_movies:
                if movie in test_movies:
                    hit += 1
                all_rec_movies.add(movie)
                popular_sum += math.log(1 + self.movie_popular[movie])
```

```
            rec_count += N
            test_count += len(test_movies)

        precision = hit / (1.0 * rec_count)
        recall = hit / (1.0 * test_count)
        coverage = len(all_rec_movies) / (1.0 * self.movie_count)
        popularity = popular_sum / (1.0 * rec_count)

        print('precision=%.4f\trecall=%.4f\tcoverage=%.4f\tpopularity=%.4f' %
              (precision, recall, coverage, popularity), file=sys.stderr)


if __name__ == '__main__':
    ratingfile = './new_data.csv'
    itemcf = ItemBasedCF()
    itemcf.generate_dataset(ratingfile)
    itemcf.calc_movie_sim()
    itemcf.evaluate()
```

Listing 1: Item-Based Collaborative Filtering

Vincent	Mongol: The Rise of Genghis Khan	Boyz n the Hood
	Western Religion	Journey to the Center of the Earth
	Re-Kill	
Edgar	Superman III	Event Horizon
	The Avengers	Double Jeopardy
	Restoration	
Addilyn	Timeline	The Work and the Glory II: American Zion
	Marmaduke	Blazing Saddles
	The Ugly Truth	
Marlee	Waltz with Bashir	Coffee Town
	Saw II	The Perfect Wave
	Frozen River	
Javier	Yu-Gi-Oh! Duel Monsters	Blazing Saddles
	Neighbors	The Ugly Truth
	Life of Pi	

We are asked to design a model to recommend movies to different users. By viewing each row of the dataset of movie reviews, we found among 2000 movies, only few of them were rated by a specific user which means the rating matrix for each user is very sparse. Due to the sparsity issue, there are huge limitations for content filtering even we have data on movie features. Therefore, we chose collaborative filtering as our method to do the recommendation. There are two classes of collaborative filtering, one is user based collaborative filtering (UserCF) and the other one is item based collaborative filtering (ItemCF). UserCF is to find the similarities among different user and gives the recommendation based on it. ItemCF finds the similarities among the items being rated. ItemCF is very suitable for occasions where the user potentially shows interests to one specific types of item with certain features and the number of users of the website is much more than the number of items. Generally speaking, the movies one user is interested in potentially have some features in common and they prefer to be recommended for movies that are similar to those they watched and were highly rated. On the other side, film websites usually have much more users than the number of movies in their dataset. Thereforefrom a very practical point of view, we chose ItemCF as our method.
The core step of ItemCF is to find the similarity between any pair of movies. Basically, if two movies are both rated by a certain user, then they will be regarded as similar to some content. In our model,

we generated a similarity matrix to refer the similarity factor of each pair of movies. For each user, taking one of movies they have rated as a reference each time, if he rated another movie, then the similarity factor plus one. We did this many times until every user's data was traversed, then we got the number of each pair of movies that indicates the times they were rated simultaneously. The final similarity factor of each pair of movies was calculated by cosine similarity. It can be represented as $cos(u, v) = \frac{u*v}{|u|*|v|}$. In this equation, u and v are 0-1 vectors of movies in each pair we are focusing on. In the vector, '0' refers to this movie is has not been rated by the user yet while '1' refers to this movie has been rated by the user already. To recommend movies to a specific user, a movie that has been watched by this user was selected as reference each time, we found the most related movies by sorting the similarity factor and give the related movies a value calculated by multiplying the similarity factor with the reference movie's rating. Then we sorted it again to find the movies we hope to recommend. This model based on ItemCF could work well. However, it always recommends movies to the user that have similarities with those they have watched before. In other words, it lack of the diversity of choice which means when the user tend to try something new, this model may not perform well[1].

Discussion

Difficulties

Assessing the quality of a recommendation system before deploying it to users is difficult, we believe there are several causes for this:
1. Precision
The core algorithm of a recommendation system is a machine learning method. Whether it is a machine learning problem such as prediction, classification, regression etc. all have their own system of evaluation metrics. The accuracy of a recommendation system can also be measured using the different machine learning paradigms to which the recommendation algorithm belongs. An inappropriate algorithm can also lead to inaccurate recommendations to users.
2. Real time and data communication
The interests of users change over time, so it is particularly important that the recommendation system adapts to the changing interests of users in real time, and that it can recommend the "target objects" that users need in near real time. Products such as news and information need to be updated in real time, and the data needs to be updated in time to make recommendations. If the recommendation system only uses older data for recommendations, then the results will be relatively ineffective.

Challenges and problems

The recommendation system faces the following challenges and problems:
1.Data Sparsity. Because of the small amount of data, there are no users who have rated the same items, so it is impossible to calculate the similarity between them (for example, in this assignment, the movie recommendation system we designed has many movies that are rated by only a small number of users, which means these movies will rarely be recommended, even if that small number of users gives them a high rating. Similarly, for those niche groups with different preferences, the system cannot find users with the same specific preferences, which also leads to worse recommendation results).
2.Gray Sheep problem. The main assumption of the collaborative filtering we used is that the target user will prefer items liked by people with similar preferences to the target user. In such a recommender system, the user's profile is represented according to their preferences for items in the system. However, there will be some people in the data whose preferences are different from anyone else's and therefore it is difficult to make recommendations to them.
3.Diversity and precision cannot be guaranteed at the same time. If company wants to recommend items to a user he likes, the best way to do this is to recommend items that are rated particularly highly through the recommendation system, as these items have a greater chance of being preferred. However, such a recommendation does not necessarily produce a good experience, because the user is likely to already know about these popular items, so the amount of information received is minimal and the user will not agree that it is a personalized recommendation.[2]

References

[1] Introduction to Recommender Systems (II): Collaborative Filtering. Zhihu. Retrieved January 31, 2023, from https://zhuanlan.zhihu.com/p/268401920

[2] C-N. Ziegler, S.M. Mcnee, J.A. Konstan, G. Lausen, Improving recommendation lists through topic diversification, in: Proceedings of the 14th International Conference on World Wide Web, ACM Press, New York, 2005, pp. 22-32.

Reflection on the previous module

Yahui Wu

Last module we mainly talked about some different applications of computing. There are mainly three categories of computing being discussed-basic computing, advanced computing and AI. Basic computing could be a time consuming method for a human but the procedure could be very easy. Advanced computing is more sophisticated while it needs certain algorithms including simulations, optimization etc. There are two main classed of AI problem. Rule-based AI needs logic, rules and human's intuition to build up the model and solve the question. If we use data to let our computer to learn and form the problem and use some statistics techniques, then it's machine learning.

Predict the temperature. The temperature prediction can be considered in two different ways. One is to predict the temperature in the short term like the temperature of tomorrow, and the other one is to predict the temperature for the same date as today but next year. To predict the temperature in the short term, we can use models based on physical condition of the weather which consist of very complicated equations such as the equations consider the wind, temperature, humidity and surface pressure. However, to predict the temperature in the long term the short-term method would fail since it accumulates error. What we could use to predict the temperature for the same date but next year is statistics method which collected data of temperature for the previous years and build the model to predict the temperature we need according to the previous data.

Bingo lottery problem. The tickets should be designed in a fair way which means the number of each combination for each ticket should be selected randomly. To solve this question, we implemented the random forest algorithm based on some assumptions. Randomized greedy search solution could also be in good use to solve this problem. The combination of the 5 numbers is formed randomly and we could calculate the number of wins based on this combination. Once the number of win is not suitable, we can randomly choose which number of the combination to change.

Public transport departure forecast In this question, we chose three features which could indicates the status of the running transport to predict the departure time. The features were used based on the equations that reflect the physical condition of a bus or train and combined with the data of previous buses. The departure time could be considered as a combination of two parts. One part is the fixed predict time which is read by the timetable or from other source of information and the other part is the correction due to some realistic situations. The information needed for the correction can be collected from GPS or previous busses etc. Meanwhile, simulation of intersections is another good choice which can save computational time.

Film festival problem In this problem, every user of the designed algorithm has a preference about the movie on their wishing list. As a result, the greedy search may not work since it could only give a solution to this problem in the end but not the optimal one and the problem needs to be turned into a search problem at first. However, there are some other method that could be helpful such as ILP, relaxation etc. By watching the record of follow-up session, we found that the schedule of this problem was described as a 0-1 vector, which is great inspiration for us to transfer complicated data into a more simplified way of descriptions.

Product rating in consumer test Basically, the rating is the linear combination of weighted data of some different aspects about the product like data from users and the features of the product itself. To summarize, learning the lectures of module one is a great opportunity for me to get further and deeper into AI. It's important to solve a realistic problem by realizing what's the essence of the problem and what's the most suitable model to describe the problem. With some practical problems in the assignment, I started to solve problems in a more artificial intelligent way.

Tianshuo Xiao

In the previous lesson, we learned mainly about the history of AI and some of the applications in the development process. Computers have different applications, some of which are like calculators and help people solve simple problems. Advanced computers include complex forms of computing that can try to replicate and simulate certain people's behavior, we call this AI. AI is not just about advanced computing, it requires logic and rules. When we are given a set of data, we can model them, extract their features, and use machine learning to evoke them. When building a model, there are many options to choose from, such as discrete models, dynamic programming, greedy algorithms, and various algorithmic tools. I use this knowledge to complete assignment 1.

Predict the temperature. In our assignments, we have found that weather and temperature forecasting now relies on numerical weather forecasting. As atmospheric movement always follows certain physical laws, people put the rules of atmospheric movement into a series of mathematical equations and then use high performance computers to calculate them to get the conditions for future weather development changes. For short-term forecasts we can use mathematical equations to make predictions. For long-term weather and temperature forecasts we extract the factors and use machine learning to make prediction.

We reflected on this question through the explanatory video.Weather conditions and temperatures mainly depend on physical equations, which include the direction of wind, humidity, continuity of mass, surface pressure, etc. Then we perform fluid calculations based on the physical laws of temperature and humidity. When predicting weather over a long period of time, there are other different factors that influence the error, in which Euler's theorem is mainly used. Such errors are unavoidable. For example, using Kepler's law to calculate the gravitational force between celestial bodies, we can use this example to extend to fluids and thus predict the weather.

Bingo lottery problem. For this problem, we made some assumptions in our assignment and permuted them to perform probability calculations. We found that the data were all discrete, so we wanted to implement a random forest algorithm to classify and extract each time so that the final winning rate would be 12%.

When we were designing this system, we needed to sufficiently consider the randomness of the process to meet the design requirements. Through the video we found that we can use randomized greedy search solution. In each extraction they only need to replace one drawer at random and then get the correct rate to see in case it goes down. If it drops, which means it is getting closer to the solution. The advantage of this method is that the strategy of changing the numbers randomly to get it closer to the winning rate, 12%.

Public transport departure forecast In the problem of predicting the departure time of public transport, the departure time can become uncontrollable as it is affected by various factors such as weather, routes etc. We considered this to be a time series related prediction and so we used a linear regression approach in the problem. The difficulty in this problem is in the collection of data. There are various sources of data, such as GPS, previous departure times, simulation data, etc. If we use a single type of data, it will lead to a large error in the predicted time. Therefore we can use a method that weights different calculations to predict public transport departure times. For example, the vehicle's position on the GPS and the time of previous vehicles through this route. We can also consider the simulation of intersection passage times to simulate the route and timing of public transport.

Film festival problem In this problem, the optimal solution is involved, so we choose a greedy algorithm for the solution. However, when planning, we only considered the number of movies that would allow the user to see the maximum number of their favorite movies, without taking into account the priority of the movies. Different films have different show times, and the same film may have multiple show times. Therefore we have to prioritize how much different people prefer different films, which can be rewarded by an algorithm that assigns points to different preferences.

There are several ways in which we can resolve this here. In terms of algorithms, we can use integer linear programming. In addition, we can use some AI tools, such as ILP, for intelligent linear pro-

gramming (IoT). Dynamic planning methods can also be used to achieve the goal.

Product rating in consumer test For this problem, we mainly consider weighting the different evaluation indicators. But a challenge here is the choice of evaluation indicators. With the video we expanded our thinking to get relevant data by users, counting the most popular brands, the most popular brands, etc. Then different characteristics were extracted, such as price, quality, noise, cleanliness, etc. We performed a linear analysis and ended up with a comprehensive evaluation method. This is what we did not consider in the assignment.

In conclusion, we need to combine with modelling, data science and AI to solve the problem. AI is an emerging engineering discipline now. I also need to learn the algorithms involved and grasp the principles of how these AI tools achieve classification and scoring so that I can use AI flexibly.

Summary of lectures

Yahui Wu

Recommender Systems
January 25, 2022
In the last lecture, we learned about recommender systems. Recommender systems are widely used by many websites and they are designed to make websites run in a better way like maximizing sales. Generally, a recommend system could just recommend the best sellers but there are better ways existing. Recommender systems try to personalize the recommendation. It's important to figure out users' favorite and we must know something about users. One way to achieve that is to collect data of users passively. The data could come from logs of users' past behaviours. Based on the logs, we can predict their future behaviours. When building a recommender system, the first thing we should know is the objeictive of this system. The objective is sometimes hard to specify, measure and optimize due to the delay and unpredictable future. Thus we use proxies to reflect our objective. In this case of building a recommender system, user ratings are helpful. The next step is to predict the ratings by a specific user and recommend the items with high ratings. If we have user ratings matrix Y, user preferences matrix Θ and product features X, a assumption could in linear form such as $Y = \Theta X$. Content filtering could be built based on this linear regression model. Parameters of this model can be found by minimizing squared error. However, in reality, most of Y is missing and not uniformly which could cause overfitting. To avoid such an issue, regularization could be helpful. Content filtering is simple to achieve but sometimes we don't have complete product features X and we ignore the fact that many users behave similarity. To improve this, we could use collaborative filtering instead. It relies on user-product data only and could give similarities to similar products. One basic and useful method to achieve collaborative filtering is martix decomposition. To find the optimal parameters of the model, we could combined it with regularization as before.

Tianshuo Xiao

Recommender Systems
January 25, 2022
This lecture introduced Recommender Systems and how to build them. Recommender Systems are very common and are mainly used for personalized advice such as advertising recommendations and purchase suggestions, for which many well-known companies make a profit.
The lesson is analyzed through an example of an online book sale. To make the most profit, bookstores are not supposed to choose the best-selling or discounted books to sell, but rather to personalize them and recommend books that users like. Because if everyone had a 60% probability of buying a book, the average sales of any book would be less than 60%. In order to predict user preferences, we can obtain data from purchase logs and build recommendation systems to solve this problem. By setting up recommendation systems, we want to make a profit, get the preferences of our users, and so on. But we also face several difficulties. Firstly, it is difficult to specify and measure something, and secondly, it is difficult to optimize the system. For example, in order to obtain more profit, it is easily to measure optimize likelihood of product. So when we select which book to sell, we need to get more positive reviews and only ask long-time users.
We can use user ratings as one of our criteria, but this does not apply for some specific products. With many user ratings, we can recommend highly rated products to potential users and increase sales. Highly rated products need to be personalized before being recommended to users.
Therefore, we need to establish a relationship function between the type of book and the rating. Firstly, built the correlation matrix. Eigenvalues are extracted for the person's name, and the X matrix is vertically for the type of book and horizontally for whether the book was purchased. y matrix is vertically for the user's name and horizontally for the rating. A user preference matrix is

built. Assuming that each user has its own linear rating function, the parameters for each user are then found by minimizing the squared error and finally finding model which minimizes the error in the observed data.
We also need to deal with data missing, over-fitting. So, we need to find parameters with lower variance and filter the content (Content filtering and Collaborative filtering). In fact, we have created three matrices, so we need to do matrix decomposition, which can establish the relationship between them. Collecting data is also a key part of building the model and is valuable for data tracking. The algorithms we use need to be appropriate to our goals and the evaluation should follow them.