

Chapter 1: Introduction and Motivation

Section 1: Introduction to this book

Welcome to this book on recommender systems! I'm so happy you're reading it. You might be wondering, is this the right book for me? Let me see if I can help. This book is ideal for you if

- You are interested in understanding the technology behind the recommendations you receive every day from [social media](#), [movie streaming sites](#) and [e-commerce stores](#). OR
- You are a [Machine Learning enthusiast](#) or a [data scientist](#) looking to enhance your skill set. OR
- You are an [entrepreneur](#) interested in building a recommender system for your own business.

So now that I have covered who you are, let's get into what I will be covering in this book.

In this course, I will teach you various types of recommender systems along with their pros and cons. By the end of this course, you will learn to design a news recommendation, product recommendation and movie recommendation system. You will be able to analyze and decide which recommendation techniques will be most appropriate to apply to your business. And you will learn to implement and evaluate your chosen technique. In the next lecture, I will give you a detailed walkthrough of the entire curriculum.

You might be thinking that how am I qualified to teach you this course? For the past 6 years I have been developing new Machine Learning algorithms for mobile processors. I have published over 20 papers in international journals and was awarded the prestigious Convergent Science Network fellowship in 2015. I hold a Ph.D. from the Nanyang Technological University in Singapore where I also work as a scientist. And I am currently collaborating with the IBM research center in Silicon Valley for programming their revolutionary neural network computer.

However, since Machine Learning is closely associated with rigorous mathematics, I know how grueling the journey to learn a new Machine Learning technique can be. When I was in your shoes, there was nobody to show me that Machine Learning can be learned in a less complex and a more intuitive way. I figured that out when I was doing research on various topics of Machine Learning such as Recommender Systems. I discovered that behind each successful Machine Learning algorithm there is a physical significance or intuition that makes it work. Mathematics is required just to validate it. And in this book I wanted to demonstrate these intuitions to you for the case of recommender systems. By reading this

book, in the long run, you will start to develop this intuitive way of thinking about Machine Learning systems.

I must say that I am really excited to teach you this subject. When I started my graduate research many years back I never knew that someday I will become an expert in this field and will get the opportunity to teach this subject to you in my special way. I am sure you and I are going to have an awesome journey.

Section 2: Book Overview

In this section, I will show you the structure of this book. This book is composed of [five chapters](#). The [first chapter](#) is the introductory chapter where I shall introduce you to this book.

In the [second chapter](#), I will properly define the goals of a recommender system and show you numerous practical examples of where these systems are found in our daily lives. Then you will jump right into the action and start understanding your first recommender system in this section itself.

In the [third chapter](#), you will move on to design more sophisticated systems and eventually learn how to develop a product recommendation system similar to Amazon. You will learn how to implement Collaborative Filtering, tackle the adverse effects of very popular items, construct and normalize the Co-occurrence matrix and leverage purchase histories for making better recommendations.

In the [fourth chapter](#), you will learn how to develop a movie recommendation system like Netflix by Matrix Factorization. You will learn how to automatically construct “topic” vectors corresponding to users and movies that will help you in predicting movie ratings. Moreover, you will discover the infamous cold-start problem and understand how to solve it by blending or combining different recommendation approaches.

After making you familiar with popular recommender systems this book will show in the [fifth chapter](#) how to evaluate the performance of these systems by metrics such as precision and recall. You will know about optimal recommenders and learn how to make the best recommender systems.

I will conclude this book in the [sixth chapter](#) and say some parting comments. Let us now jump in the world of recommender systems.

Chapter 2: Introduction to Recommender Systems

In this chapter, you will get an overview of recommender systems. Moreover, you will see the popular platforms where these systems are typically used. Furthermore, you will learn two methods namely the popularity and classification approaches of designing recommender systems.

Section 2: Recommender Systems Overview

In the previous section, you learned what this book is about which is to understand recommender systems. In this section, I am going to talk a bit about these systems. One prototypical example of where recommender systems are really useful is in recommending products. Imagine that you have a large set of products and there are some users, and you want to recommend a subset of your products to those users.

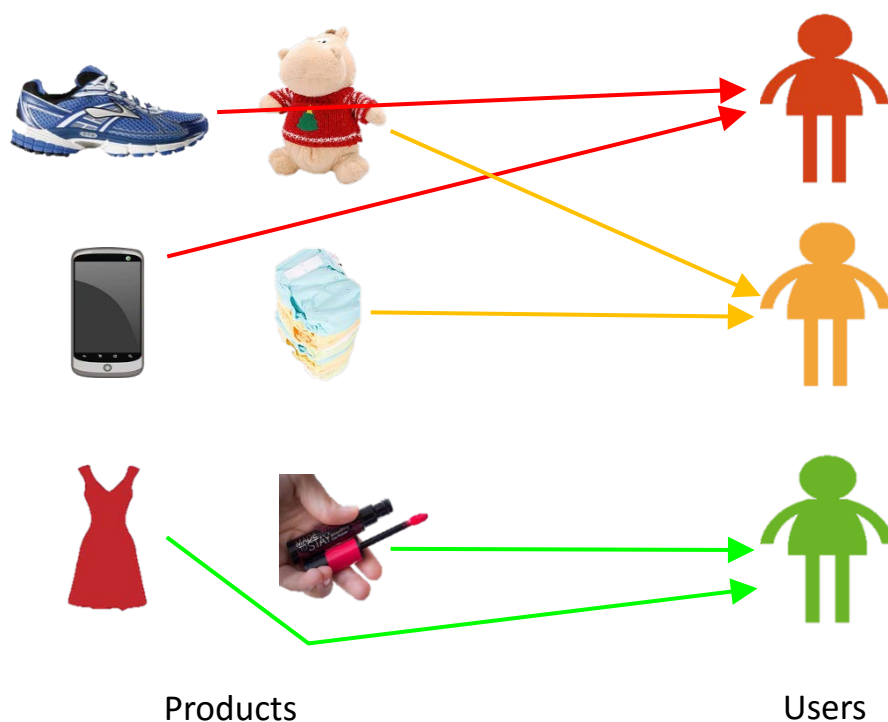


Figure 1: A cartoon diagram showing the set of products and users for an example recommender system. Each user likes a subset of the available products.

Well the question is - how are you going to do this? And what I am going to talk about in this book is how to use **Machine Learning** techniques in order to use the past history of your own purchases, as well as the purchases of other people, to determine which products to recommend to you. Recommender

systems just have a really, really wide range of applications and they've exploded in popularity over the last decade. But Amazon was an early pioneer in this area, focused on an application we're going to talk about in this course, which is product recommendation.

[Your Amazon.com](#) > **Recommended for You**
(If you're not Subhrajit Roy, click [here](#).)

These recommendations are based on [items you own](#) and more.

view: **All** | [New Releases](#) | [Coming Soon](#)

- 

AmazonBasics 11.6-Inch Laptop Sleeve
by AmazonBasics (July 4, 2013)
Average Customer Review: ★★★★★ (5,498)
In Stock

List Price: \$9.99
Price: \$9.84

☐ I own it ☐ Not interested ☒ ★★★★★ Rate this item

Recommended because you added **ASUS X205TA 11.6 Inch Laptop** to your Shopping Cart ([Fix this](#))

Figure 2: An example showing that Amazon, a pioneer in product recommendations, is recommending an item based on your shopping cart.

Another example that really popularized recommender systems was a competition that Netflix ran starting back in 2006 and ending in 2009, where there was a \$1 million prize for the person who provided the best recommender system for recommending movies to users. We will talk a little bit about that application in this course. It really spurred a lot of activity in this area, developing new recommender systems and a lot of research following that.



Figure 3: The winning moment! This picture shows the group that received the 1 million USD Netflix prize.

Section 3: Recommender Systems in our daily lives

In the previous few sections, I introduced the concept of recommender systems to you. In this section, I shall discuss some areas in which recommender systems are playing a really active role behind the scenes. We are going to see in this discussion that depending on the specific application, different aspects of the objective we're trying to optimize are going to be important.



Figure 4: Some examples where you see recommender systems in use every day.

But before we start talking about recommenders systems, it's really important to talk about the idea of [personalization](#), because personalization is transforming our experience of the world. In today's world there are lots and lots of articles out there. There are numerous webpages in the internet. And we can't possibly be expected to browse all of them. Another example would be YouTube. It's quoted that in YouTube there are roughly 100 hours of video uploaded per minute. And the question is what videos do you care about? You know that when you go to YouTube you want to watch some videos that are of interest to you. So this is a clear example of information overload. There's just way too much content there for you to possibly be able to go and find what you're interested in without somebody helping that content come to you. So browsing i.e. [the traditional form of browsing is now history](#). There needs to be some way in which the content that's relevant to you is automatically discovered. This is the notion of personalization, where we want to connect users like yourself, who goes to YouTube and items which, in this case, are going to be videos on YouTube. Personalization is going to play a key role in this notion of recommender systems.

Let us now talk about some examples where recommender systems are very important. One very classic example that I mentioned in the previous lecture was of the streaming website Netflix where people go and watch videos. Netflix has this goal of trying to make suggestions of which movies or TV shows might be of interest to the person who has come to the site. What I am interested in is to teach you how to design such systems.

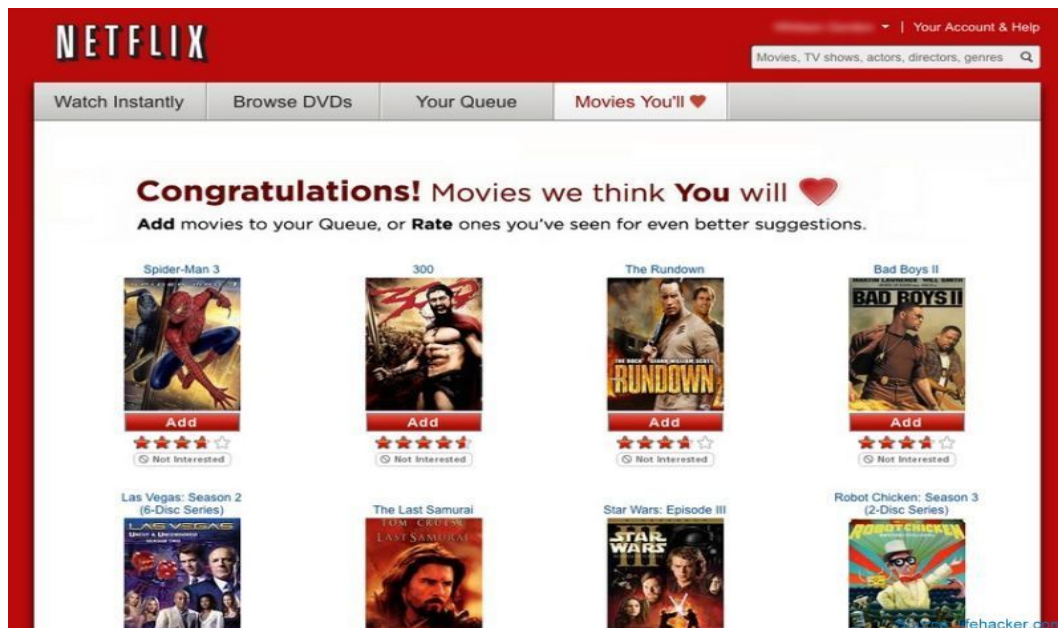


Figure 5: Netflix recommending some movies and TV shows to me based on my previous ratings.

Another example, again going back to something I talked about earlier, is [Amazon making product recommendations](#). So you go to Amazon and you purchase or browse some products, you will see on the site that it makes a recommendation of other products you might be interested in. But one important thing to keep in mind is that while making recommendations you not only have to take into account the interest that the buyer had in this one session but also in previous few sessions. For example, Figure 6 below shows that since I added an 11.6 inch ASUS laptop to my shopping cart in a session, Amazon is recommending me an 11.6 inch laptop sleeve. However, I'm not always looking for accessories for my new laptop. There are lots of other reasons why I go to Amazon and make purchases. And if you look at my history of purchases you're going to make much better recommendations for me than just based on a single session. Moreover, recommendations might change over time. So there's interest in making recommendations, for example, about what I might be interested in purchasing today. So if you look at my purchase history, a year ago I was buying a lot of furniture since I moved to a

new apartment. But that's probably not something that I'm very likely to purchase today. So the recommendations that Amazon presents to me today have to adapt with time as shown in Figure 7.

[Your Amazon.com](#) > **Recommended for You**
(If you're not Subhrajit Roy, click here.)

These recommendations are based on [items you own](#) and more.

view: **All** | [New Releases](#) | [Coming Soon](#)

- 

AmazonBasics 11.6-Inch Laptop Sleeve
by AmazonBasics (July 4, 2013)
Average Customer Review: ★★★★★ (5,498)
In Stock

List Price: \$9.99
Price: \$9.84


☐ I own it ☐ Not interested ☒ ★★★★★ Rate this item

Recommended because you added **ASUS X205TA 11.6 Inch Laptop** to your Shopping Cart ([Fix this](#))


Figure 6: Since I added an 11.6 inch ASUS laptop to my cart, Amazon is recommending me to buy an 11.6 inch laptop sleeve.

Today's Recommendations For You


Here's a daily sample of items recommended for you. Click here to [see all recommendations](#)



Even Faster Web Sites: Performance... (Paperback) by Steve Souders
★★★★★ (7) \$23.10
[Fix this recommendation](#)



Simply JavaScript (Paperback) by Kevin Yank
★★★★☆ (19) \$26.37
[Fix this recommendation](#)



The Art & Science of JavaScript (Paperback)
★★★★★ (5)
[Fix this recommendation](#)

Any Category Algorithms Boxed Sets Business & Culture Java
Graphic Design Microsoft Networking Networks, Protocols & APIs New SQL

Source: www.webdesignerdepot.com

Figure 7: The recommendations that Amazon makes adapt with time. Hence, here I am getting a today's recommendations.

Just as on demand video with personalized recommendation has really revolutionized how people watch movies and TV shows, likewise there are many websites that provide [streaming audio with personal recommendations](#). However, unlike on demand video, in this case I want one song to play after another. And what I want is to listen to a coherent stream of songs. I want to hear songs similar to what I like. I don't want rapid switching happening between songs of very different genres. For example I don't want to hear a heavy metal song after a classical one. At the same time I don't want a song I just heard to play again and again and again. So I want some sense of recommendations that are coherent but I also want them to provide a diverse sequence of songs to listen to.

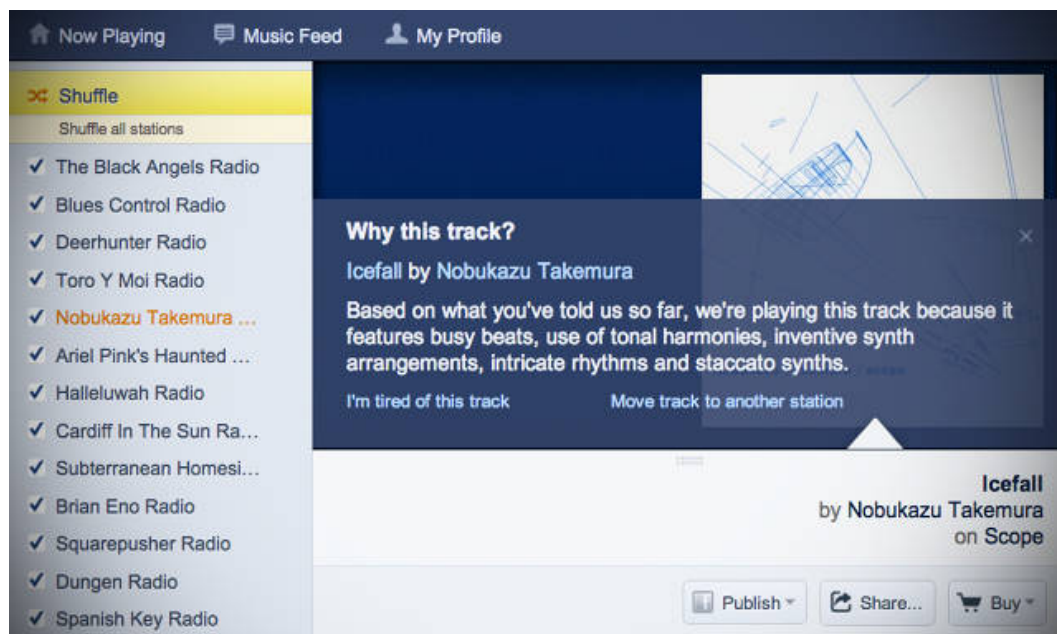


Figure 8: Pandora recommending a coherent and diverse stream of songs.

Another critical area where recommender systems have played a very active role is in [social networking](#). For example, Facebook has tons and tons of users. And the system wants to form connections between these users. Let us consider that the graph in Figure 9a represents the connections between users on Facebook and maybe you are this node right here (Figure 9b), and Facebook wants to recommend other people you might be interested in connecting with. It is important to note that in this application both the users and the quote unquote items are of the same type. You are both people. So when you are a

user on Facebook the things that are being recommended to you i.e. the items themselves are other people. So, users and items are exactly of the same type in this application.



Figure 9: Facebook recommends to you other people you might be interested in connecting with.

Now, you and I have reached a point in this section where I shall shift gears. Till now the recommendation systems that I have talked about have really focused on online media. In recent years, more and more people are finding out other areas in which recommender systems can play a very important role. One pioneer example I can think of is called “[drug-target interactions](#)”. Here drugs that have been studied in detail are considered, for example, let's talk about aspirin. It's been well studied as a treatment for headaches. But what if it's discovered to have some other possible use? For example, for blood thinning, for heart patients, etc.? If you can find these types of relationships i.e. if you can repurpose this drug for some other treatment, then that could be really useful since it's really a quite costly and lengthy process to get FDA approval for a completely new drug. If you can take a drug where the types of side effects and the possible risks associated are already well known and well-studied, then it's a lot easier to get approval for treatment with some other condition. So this is a case where you might say, if I like Aspirin for headaches, I might also like Aspirin for my heart condition and this is where recommender systems come in. Hence, these systems are playing an active role in [medicine](#) as well. In this section, I showed you the diversity of applications where we see recommender systems in use.

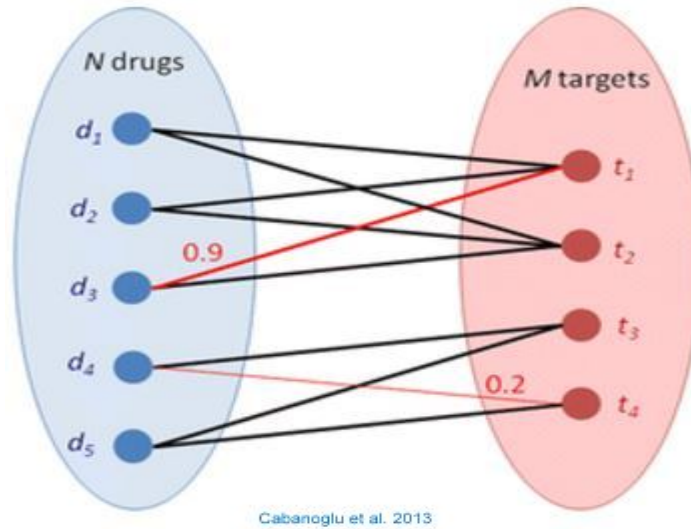


Figure 10: Recommender systems are not only confined to online media, but also used in the medical industry.

Section 4: Building a recommender system

In the previous section, I talked about a lot of application domains where you see recommender systems. Now let's talk about how you can actually build such a recommender system. There are numerous approaches for implementing these types of systems and I will discuss some of them in this lecture.

The first method which I refer to as Method 0 is recommending products just based on the popularity of each product. Although this is a very simple approach, it is actually really popular with news sites. For example, New York Times has this list on their website which shows the most popular articles. It is sorted by different topics, for example, trending articles or the most watched news or the most emailed articles and so on. Here when we think about recommending articles to other readers, we're just sorting the articles by how often they were shared or read by other readers on New York Times. So this can work fairly well. It is very much possible that you might actually discover an article that you're interested in reading by using this type of approach.

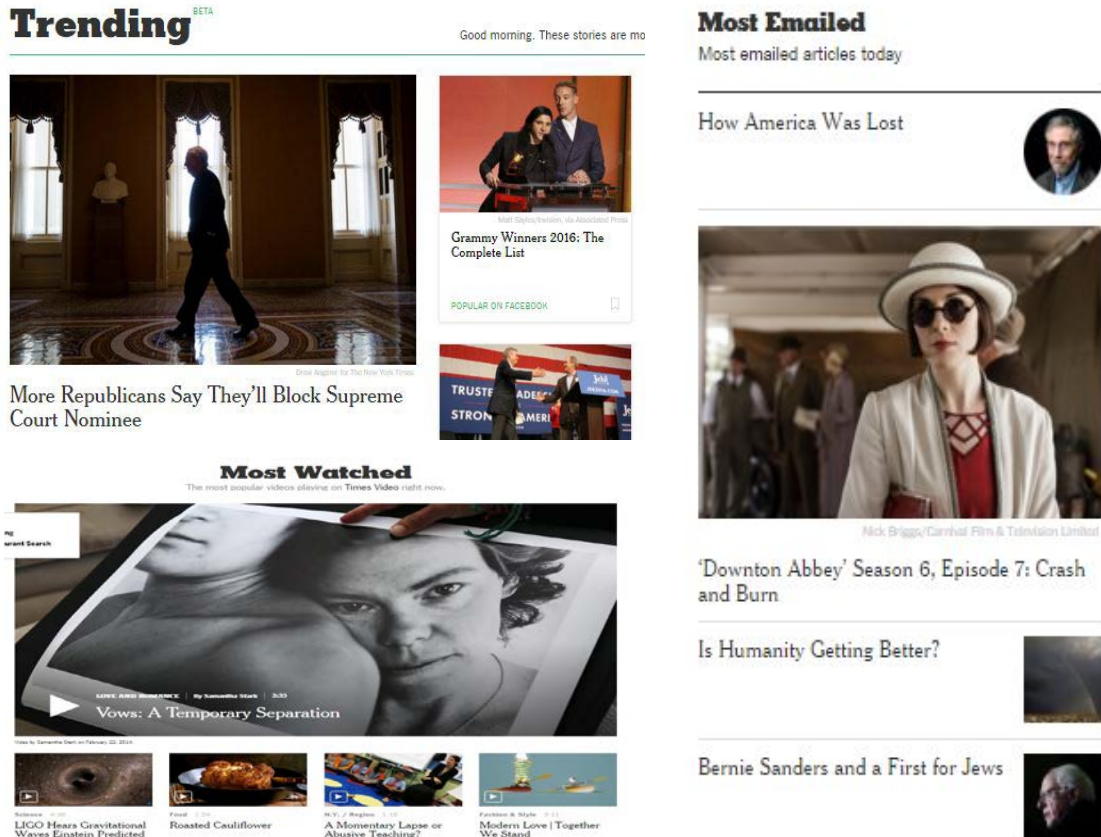


Figure 11: New York Times has implemented few popularity based recommender systems in their website.

However, one disappointing thing about Method 0 is that it completely lacks personalization since all the recommendations are based on the entire population of New York Times readers out there, which is actually pretty diverse. So instead, I would, love to have a method that knew a little bit more about me and my interests before recommending news articles to me. This thought leads to Method 1 of recommending products.

In Method 1 I will try to add a little bit of personalization using a model where you are going to use features of both the products and the users to make our recommendations. This model is used to classify whether a person likes or does not like a product and is known as the classification model. This classification model is going to take in features about the user, features about the past purchases of that user, features about the product that we're thinking about possibly recommending, as well as potentially lots of other features that we can think about. And you are going to shove all these features

into our classification model, and it's either going to spit out, yes, I believe this person is going to like this product or, no, I don't think they're going to like it as shown in Figure 12.

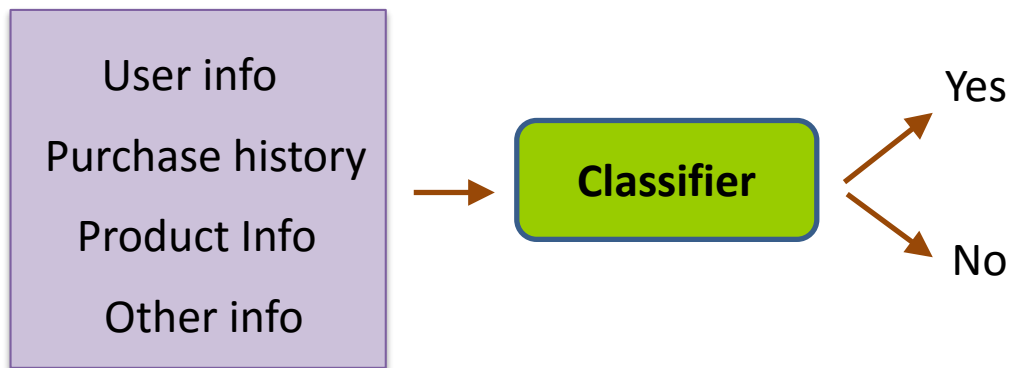


Figure 12: The classifier model of recommender systems.

Note that this type of classification approach has a lot of things going for it. First, it can be very personalized. You are using features of the user as well as the user's purchase history. In addition, this model can capture context. For example, you can take into consideration what the time of the day, what the person just saw, etc. Like maybe I'm much more likely to purchase a textbook during the day or home products at night. Another really nice thing about this approach is the fact that it can handle a very limited user history. For example, let's say I'm a user on Amazon and I haven't purchased many items in the past. So for a lot of approaches, there might be a lot of ambiguity about what I might be interested in or not, but if you have information about me, such as my age, that alone can be very predictive of what I might like or not like.

This type of classification approach has limitations as well. And one of the important limitations is the fact that the features that I am talking about that can be so informative about the products that we're going to potentially like or not might not be available. For example, you might not know the user's age. You might not know his/her gender. And the list goes on and on. And, likewise, for a product, you might also have either missing information or very poorly written product descriptions, especially on Amazon where there are lots of people selling different products. The quality of that information might be pretty low. So often what you actually see is a method called collaborative filtering which works better than this type of classification approach. I am going to talk about it in detail in the next chapter.

Chapter 3: Collaborative Filtering and Co-occurrence matrix

In this chapter, you will learn a powerful recommendation technique termed as Collaborative Filtering. This method is being used by various companies around the globe for numerous recommendation tasks. However, in this book I will explain this system to you in connection to a task that you see every day. That is the product recommendation task and the pioneer company in this domain is Amazon.

Section 1: Collaborative Filtering: People who bought this also bought...

In the previous section, you looked into the areas where recommender systems are typically used. Moreover, you saw two simple ways of building recommender systems based on popularity and classification.

In this section, I will tell you about the concept of **collaborative filtering**. The idea is that somehow I want to leverage what other people have purchased to make product recommendations to you. It seems very intuitive that when I am thinking about doing product recommendation, I want to build in information like if a person bought this item, then they're probably also interested in some other item because I have seen lots and lots of examples in the past of people buying those pairs of items together. Maybe not simultaneously at the same time, but in the course of their purchase histories. This brings us to the idea of co-occurrence of purchases. For example, if I was just on Amazon, buying spoons for my kitchen, and since Amazon has probably seen lots of examples of people who bought spoons also bought forks, it will recommend me to buy forks. The question is - how can you use this type of co-occurrence of purchases to make recommendations? Remember that although we are talking about product recommendations here, this approach is not limited to this domain and can be used in any area where recommendations are needed.

I will now talk about this co-occurrence matrix that we are going to build up. And this matrix is going to store all of the information about which items people bought together. Remember that when I say together I don't mean simultaneously, but just together at some point in their history of purchases. Now, I am going to build up a Matrix C which is an items by items matrix. We are going to list all the different items for our rows of this matrix. And likewise, all the different items, for the columns. So, for example, consider that this row of this matrix corresponds to spoons. Then if this row is let's say the third row of this matrix, then the third column of this matrix also corresponds to spoons as well. Since it is known that many people purchased both spoons and forks, let's look at the row for spoons and then I

scroll over to the column for forks which is say this column right here. And in this entry, of this matrix there is some number entered. This number represents the number of people who purchased both spoons and forks. Now, let me ask you a quick question. Is the number of people who purchased spoons and forks the same as the number of people who purchased forks and spoons?

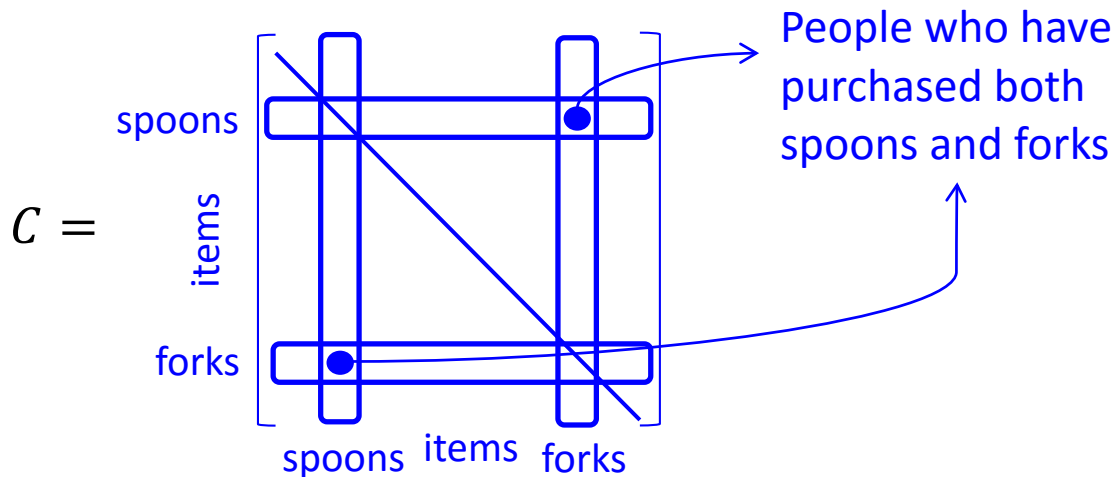


Figure 1: The co-occurrence matrix C that stores the counts for the simultaneous buying of any two items.

Yes, of course. So if you go to the forks row and then scroll over to the spoons column you will notice that this is the exact same number. This revelation has a very nice implication. It means that this matrix here is a symmetric matrix that i.e. if you look across the diagonal here, then you are going to see a reflection. If you took this matrix and folded it across the shown diagonal line you would get exactly the same numbers matching up.

Let me just reiterate the way you are going to build up this co-occurrence matrix. You have to search through all the user's history of purchases they've made, and count. Every time you see a purchase of spoons you must add this buying of spoons along with all the other items i.e. you must add one to that entries, and you should keep incrementing the matrix while searching over users.

Now I will show how you can use this co-occurrence matrix in order to make recommendations. And it's really, really straight forward. So let's say that a user has just purchased spoons, and you want to make some recommendation for him/her. Well what you must do is that you must look at the spoons row of this matrix. So, you go back to the C matrix you created and fetch the spoons row out of the matrix. This row will look something like this. This row indicates how many times people have bought spoons with

other items. Remember that forks were one of the items and let us say that there are 150 counts of forks. And then there's also let's say plates. And maybe there are 43 counts of plates. Let's think of something else. Let's say that there's some gaming CD and let's say there are no counts of that gaming CD. You are going to have this whole vector of counts of how many times people who bought spoons bought all these other products. Using this, now you can make your product recommendations in a very straight-forward way. All you have to do is that you have to sort this vector and recommend the items with the largest counts. So maybe, you will recommend forks, plates, trays, glasses and similar things for somebody who just purchased spoons.

Section 2: Adverse effect of very popular items

In the previous section, you looked into the concept of collaborative filtering. You also found out how to create and read co-occurrence matrices for product recommendations.

In this section, I will show you one problem which collaborative filtering faces. This problem occurs when you have hugely selling items in the co-occurrence matrix. It is tackled by normalizing the matrix and we will talk about this problem and its solution in this video and the next. Let us now talk about what happens if we have a very popular item. One of the most popular items for kitchens is spoons. There are lots of these selling in Amazon. These are, of course, an absolute must have for kitchens. So basically, if you're going to purchase any kitchenware on Amazon, you're also very likely at some point to have purchased spoons. But then let's look at some other item. Say let us look into silicone spatulas which are used for cooking and baking. It is also one of the most bought kitchenware on Amazon. Let's say that I have just bought a silicone spatula set and you want to make recommendations for me. As I discussed in the previous video, you should look into the row in the co-occurrence matrix corresponding to spatulas. Let's see what you will find in the counts vector for silicone spatula. And again you have that gaming CD that nobody's purchasing, then you have spoons, then you have plates, then you have forks, and all the other products you can imagine. Since spoons are highly popular items, it has some enormous number. Let's say 2 million. Seeing this huge number you will recommend this item.



Venice spoons

Oxo spatulas

Figure 2: Since Venice spoons are extremely popular items in Amazon, they affect the recommendation system when you try to buy any other kitchen item.

However, what ends up happening is that regardless of what product I'm looking at or whatever I have purchased - be it spatulas or bowls or a mixer the system always recommends me to buy spoons. Thus having very large counts for popular items drowns out all the other effects. And so, the system is just going back to the simple case of recommending based on popularity of items, which is what we were trying to address in the first place by designing sophisticated recommender systems. So let's think about how to make the recommendations a little bit more personalized and have the effect that everybody buying spoons doesn't mean that that's what I'm particularly interested in when I just bought spatulas. In the next video I will talk about how to handle this.

Section 3: Normalizing the Co-occurrence matrix

In the previous video, you looked into a problem that collaborative filtering faces when there are very popular items. This problem can be solved by normalizing the co-occurrence matrix. In this video I will explain this technique to you.

As I said, to handle this situation of having very popular items, you will have to normalize the co-occurrence matrix. And one way in which you can normalize this matrix is with something called Jaccard similarity. By using Jaccard similarity you can account for very popular items and the way it works is pretty intuitive. You have to count the number of people who purchased some item i and some item j as shown in Figure . First, you have to count the number of people who purchased both i and j . This is the count that the co-occurrence matrix had before. So these are the raw counts. Now you must normalize this entry by the number of people who purchased either of these items i or j . Let me explain this to you by using a simple Venn diagram. Consider that the orange shaded region in the left is the world of people that purchased item i . And the orange shaded region in the right represents the world of people who purchased item j . And here, in this shaded area in middle, are the people who purchased both items i and j . So what you are going to do is that you will read the count of this shaded area which now becomes the numerator. And you will normalize i.e. divide this by the total area of this diagram. Note that the total area represents the entire world of unique users that purchased items i or j . And that

becomes the denominator. This technique is one way of normalizing the co-occurrence matrix. There are other methods such as [cosine similarity](#) that is described below.

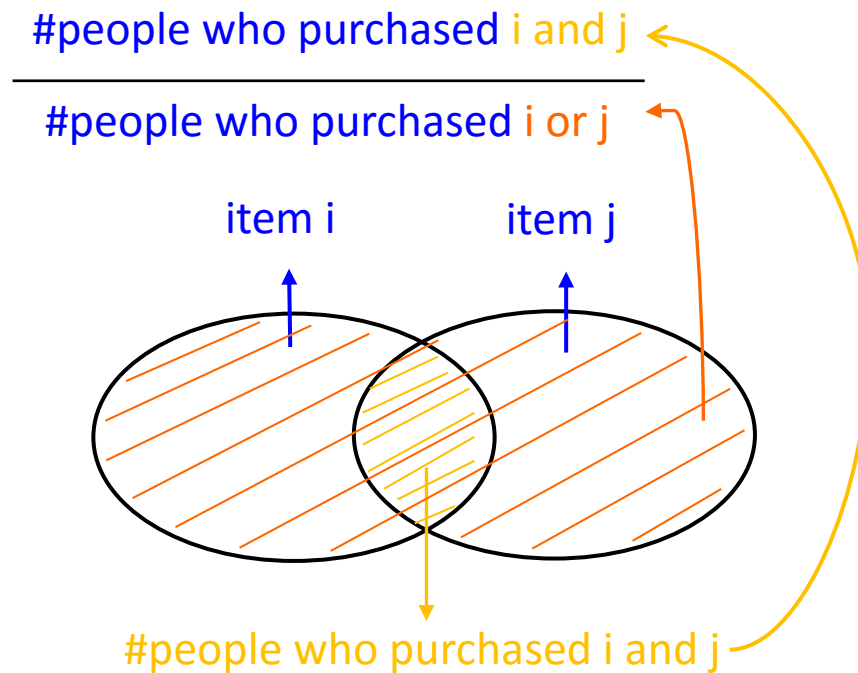


Figure 3: The Venn diagram describing the way by which Jaccard similarity can be used to normalize products.

[Cosine similarity](#) is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in $[0,1]$.

Note that these bounds apply for any number of dimensions, and cosine similarity is most commonly used in high-dimensional positive spaces. For example, in information retrieval and text mining, each term is notionally assigned a different dimension and a document is characterized by a vector where the value of each dimension corresponds to the number of times that term appears in the document. Cosine similarity then gives a useful measure of how similar two documents are likely to be in terms of their subject matter. The technique is also used to measure cohesion within clusters in the field of data mining.

Cosine distance is a term often used for the complement in positive space, that is: $D_C(A, B) = 1 - S_C(A, B)$. It is important to note, however, that this is not a proper distance metric as it does not have the triangle inequality property and it violates the coincidence axiom; to repair the triangle inequality property while maintaining the same ordering, it is necessary to convert to angular distance.

One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors, as only the non-zero dimensions need to be considered.

Section 4: Leveraging purchase histories

Till now in this chapter, you have learned about collaborative filtering and co-occurrence matrices. You have also seen a problem that collaborative filtering typically faces when there are very popular items. After that, you saw a technique termed as normalization that is used to solve this problem.

Although the described approach typically works well in most cases, but this method has its own limitation. Here, one issue is the fact that only the current page matters i.e. when you are looking to make recommendations for me, the only thing that matters is that I just bought silicone spatula. You are not looking at the entire history of things that I've purchased to inform these recommendations. I will now talk about a way in which you can modify your approach to account for my history of purchases.

A really simple way is just to do a weighted average over the scores I would have placed on the products for each item in my purchase history. Let me give a concrete example of this. Imagine that the only items I ever purchased on Amazon were spoons and vegetable peelers. Now your job is to make recommendations for me, i.e. the user that only purchases spoons and peelers. You should go through every item that you might think about recommending, and compute the score of each item as follows. Let's say you are deciding whether to recommend forks to me. And in this case, what you should do is that you should compute a weighted average over how much you would've recommended forks just based on having purchased spoons before. You will do this by using the techniques we talked about in the previous videos i.e. by fetching the row for spoons from the co-occurrence matrix, and looking at how many times people also bought forks. But then you should also look at the row for peelers, and find out how many times people who bought peelers bought forks. Next, you should average these two results to say how much, or how likely it is that I would purchase forks, given this purchase history that I

have. Then, you should repeat this method and find the weighted average score for all the products you are thinking to recommend. Okay, so when you want to make the recommendation you just sort this weighted average scores and recommend the products that have the most score. It is very similar to the approach I talked about before, but now you are combining weights based on my purchase history.

$$Score(user, forks) = \frac{1}{2} (S_{forks\ spoons} + S_{forks\ peelers})$$

This was the simplest way to leverage on purchase histories. You could also do other variants of this method. For example, instead of doing a simple weighted average you could give more weightage to my recent purchase history to account for context and so on.

I will now talk about some limitations of this method. These are listed below:

- **First**, it doesn't use context, like time of day, at least not directly.
- **Second**, it doesn't use features of the user like my age or gender or anything like that, since it's grouping all users together when it is looking at this co-occurrence matrix.
- **Third**, it doesn't use features of the products. Everything in the inventory is just pooled together without any kind of notion of different properties of these products or users to drive these recommendations.
- **Fourth**, another problem that people face while using this method is something called the **Cold start problem**. And this is a really important problem that is encountered in a lot of different domains. But let's talk about it in this context. So, the Cold start problem occurs when we get a new user or a new product. The question is - How do you form recommendations in this case since you have no observations ever for that product. You do not have an idea about how often it's been purchased along with something else because it has never been purchased. And likewise for a user, you have no information about his/her past purchases.

Chapter 4: Matrix Factorization

In the previous chapter, you learned about collaborative filtering through co-occurrence matrix. You understood how this approach can be used to provide product recommendations like Amazon. At the end of the previous section, you looked into its limitations. In this section, you will see another recommendation technique termed as Matrix factorization that is capable of eliminating these problems. This technique is used as the brain of many recommender systems. However, the application that made this technique famous is the movie recommendation task performed by Netflix. Hence, I will explain this technique to you by teaching you how to do movie recommendations like Netflix.

Section 1: Matrix completion: The movie recommendation task

In the co-occurrence approaches I have been talking about so far, there's been no notion of different aspects of me as a user or features of the product that are driving the recommendations made. Instead I was simply counting co-occurrences of purchases and checking user histories. So a natural question that comes to mind is that whether you can somehow use some aspects of who I am and what a product is to drive the recommendations. Note that this is similar to what I talked about in the classification approach in Section 1 where some set of features of the user and the product were needed. But unlike that approach, here, you will be able to learn these features from the data itself. That will help in coping with the problems that I talked about i.e. if the features are not available. In addition, you will take into account the interactions between users and items just like you saw in the co-occurrence approach. Hence, in this technique you will combine both Method 1 i.e. the classification approach and Method 2 i.e. the co-occurrence approach.

I am going to discuss this technique in the context of a [movie recommendation task](#) because it's very intuitive to talk about this application for the methods that I am going to describe. The data we have in this application is a very big table where we have a whole bunch of users that are watching some set of movies, and they rate those movies. The table looks something like as shown in Fig. 1 below.



















User	Movie	Rating
		★ ★ ☆ ☆ ☆
		★ ★ ★ ★ ☆
		★ ★ ★ ☆ ☆
		★ ☆ ☆ ☆ ☆
		★ ★ ★ ☆ ☆
		★ ★ ★ ★ ☆
		★ ★ ★ ☆ ☆
		★ ★ ★ ★ ★
		★ ★ ★ ★ ☆

Figure 1: This figure shows the ratings given by three users to the movies they have watched.

The orange user watched three different movies and gave them a rating of two stars, four stars, and three stars respectively. And then there was this green user who watched two different movies and rated them, and this red user who rated four different movies. Eventually you will have a very big table of the user and movie ratings combinations. However, each user only watches a few of the available movies. So I am going to transform this data table into a really big - users by movies matrix of ratings. The reason it's a really big matrix is that in general there will be a lot of users and a lot of movies. But at the same time, this matrix is very sparse. This is because there are lots and lots of movies and even though there are a lot of users, there are only a few movies that any given user has watched. Now, look at this, users by movies matrix and as an example consider that this row here represents user u and this column here represents movie v . And this white square here represents the rating that user u gave to movie v . In this matrix, you will find that are few of these white squares, but there are lots and lots of blue squares. Each blue square represents a question mark. It's a case where a user has not watched a

movie or has not provided a rating for that movie. Remember that all of these blue squares represent unknown ratings. They do not represent ratings of zero. It's not that the user did not like that movie. It's just that we don't know what the user thinks about the movie. I will denote the rating that user u gave to movie v by $Rating(u, v)$ which is known for the white cells and unknown for the blue cells as shown in Fig. 2.

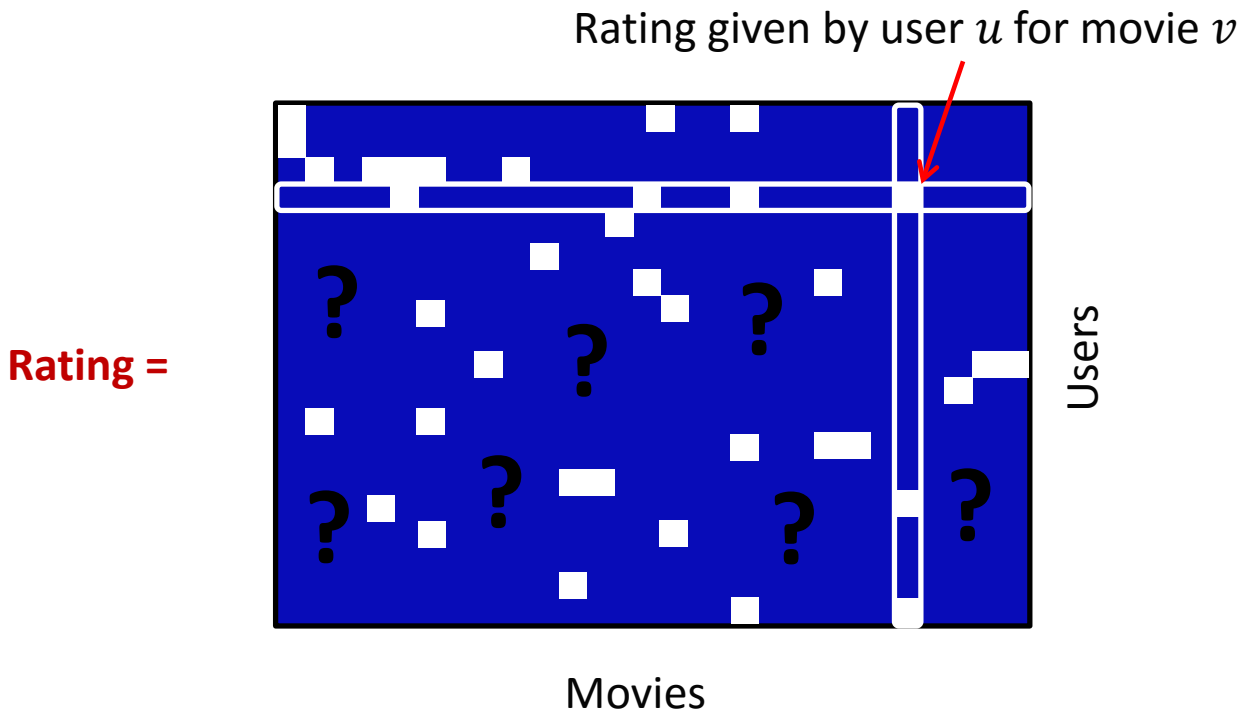


Figure 2: This figure shows the Rating matrix that stores the available ratings for all users and all movies. The known ratings are denoted by white squares and the unknown ratings are denoted by blue squares.

After creating this matrix, your goal is to fill in all these question marks i.e. all these blue cells. Consider that first you are interested in filling the unknown entries for user u . The way you are going to do this is by using all the ratings that the user has provided i.e. all the white squares for user u . Moreover, you will not only use user u 's known ratings but also use the ratings provided by other users as well. Hence, each user's unknown entries will be filled by you while using all users known ratings. In the next few sections, I will teach you how to fill the unknown entries of this really big sparse matrix.

Section 2: Recommendations from known users or items features

In the previous section, I introduced you to the movie recommendation task and showed you how to construct a matrix that stores all the known and unknown ratings. In this section, I will give you an initial idea about how you are going to make movie recommendations by using this data. I will show you how you can guess what rating a person would give to a movie that they've never watched. To do this, imagine for a moment that I have some set of features about each movie and each user. For example, I know that movie v , which, in this case, is Forrest Gump (Figure 3), is about some set of different genres, like action, and romance and drama, etc. And I have a vector that keeps track about how much the movie Forrest Gump belongs to each genre. For example, the vector here says that it is 0.2 about action; it's 0.8 about romance; it's 1.3 about drama and so on. Since I have a set of these genres, I know what the movie is about. And likewise, for every user, like this green user, I know which of these different genres the user likes and to what extent. For this green user, I have a vector that says that he/she somewhat likes action, really does not like romance, really likes drama and so on.



Figure 3: This figure shows the movie v i.e. Forrest Gump.

I am going to call this first vector i.e. the movie vector, R_v . Moreover, I am going to call this user specific vector as L_u . Now the question is – if I give these vectors to you what would you do to make a prediction of a rating? Well, one thing that might make sense is to take this movie vector and this user vector and see how much they agree. If they agree a lot, then you would guess that the user would rate that movie very highly. If they don't agree a lot then you're going to say that it's probably very likely that they will not like that movie and give it a low rating. So you are going to estimate the rating given by $\widehat{Rating}(u, v)$. That's why I put this hat over it to denote that this is not the actual rating but an estimate of how much the user, u , is going to like some movie v that they've never seen before. The method you will use to find whether these vectors agree or not, is to calculate the similarity among these vectors. To do that you should take these two vectors and multiply them element wise. So this is our R_v and this is our L_u . Now, you should multiply R_v and L_u which comes out to be 0.2 multiplied by 0.7 plus 0.8

multiplied by 0 plus 1.3 multiplied by 2.1 and so on. Assume that this sum ends up being some number like 7.3. In this case, the user vector somewhat agreed with the movie vector. Let me show you what happens if the user vector really disagreed with what the movie was. Consider another user u' and the vector corresponding to this user is $L_{u'}$ pronounced as Lu-prime. According to $L_{u'}$ this user really loves action, hates romance, hates drama, and so on. Well here, the score is going to be much lower. What you are going to get is 0.2 multiplied by 2.9 plus 0.8 multiplied by 0.01 plus 1.3 multiplied by 0.02 and so on. Since the numbers do not agree with each other, this sum would come out to be some small number like 0.97. The point you should remember is that when the movie vector and the user vector agree a lot, you will get a much larger number than when they don't. This means that you will estimate a much larger rating when these vectors agree than in the case where they disagree. For recommending movies to a user you have to apply this technique for all the movies i.e. you will multiply all the movie vectors to this user vector one by one and calculate the predicted rating for each movie. Next you have to sort the movies based on their predicted rating and then you recommend those with the largest ratings.

Before finishing this section, I want to highlight one thing here. I think you remember the rating scale was between one and five or rather zero and five since you could provide no stars if you really hated a movie. But the maximum score was a five, but note here that one of the predictions is 7.3 which is clearly greater than five. Keep in mind that with this type of model you are not restricted to the rating five. There's nothing enforcing that you have to stay within a score of zero to five. However, you can still use this to make recommendations since you are just recommending the movies with the largest scores even though those scores aren't necessarily representative of exactly how many stars a movie would get.

Section 3: Predictions in the matrix form

In the previous section, I showed you how you can use known features of users and movies for recommending movies to them. Let me take those ratings that we were talking about in the last section. But now, instead of talking about them for a specific combination of a movie and a user, I will talk about how you can represent the predictions over the entire set of users and movies. And to do this, you have to use a little bit of linear algebra.

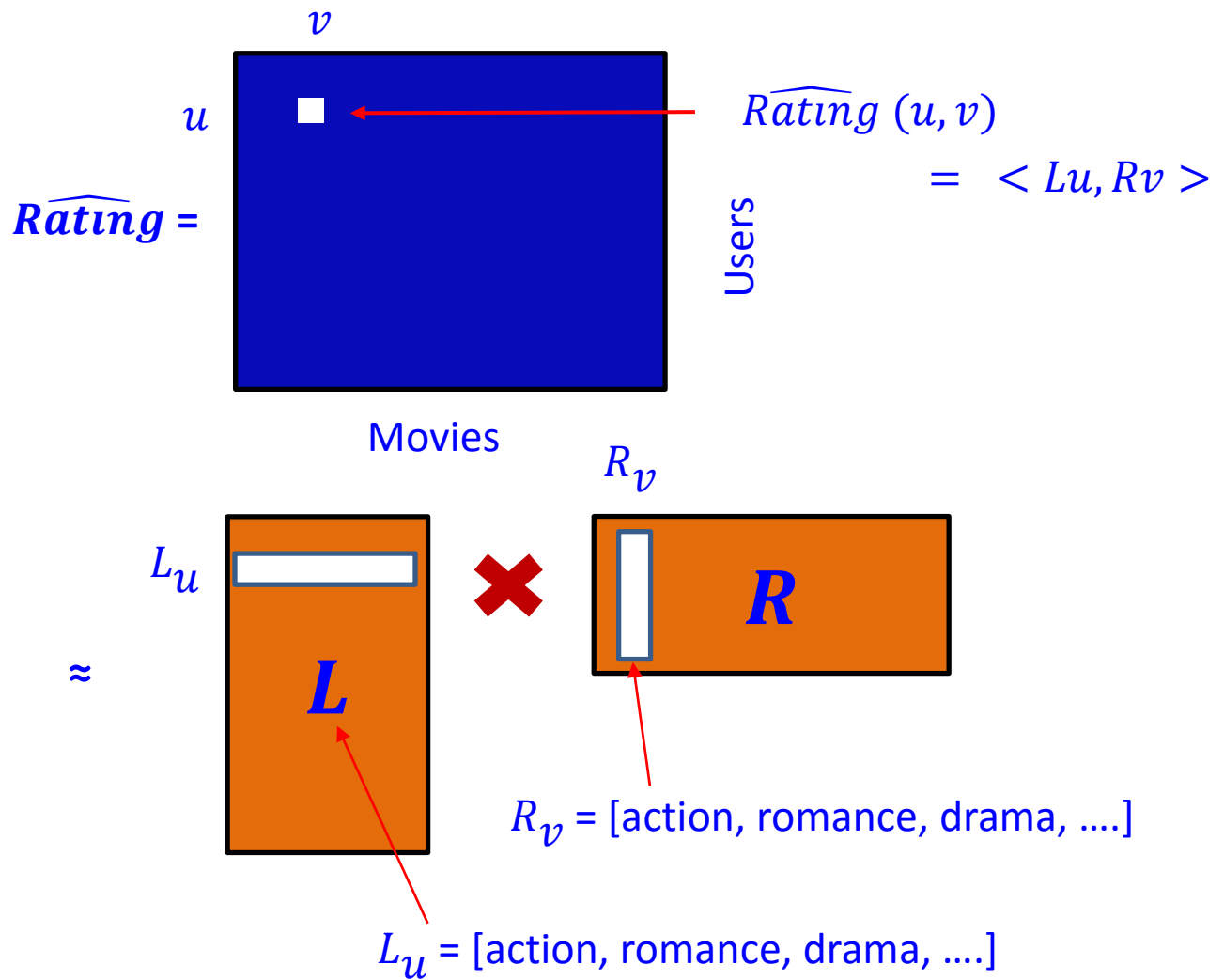


Figure 4: The user vector L_u and the movie vector R_v are part of the big matrices L and R .

In particular, in the previous chapter (also shown in Figure 4), you were looking at the score that was given to a specific movie, v , for a user, u . So this is the rating predicted for user u and movie v . Do you remember how you calculated this earlier? What you did was that you took the user vector L_u , and the movie vector R_v , and performed an element-wise product. Then you summed the elements of that product to get the rating. I am going to denote that with this little notation as shown in Figure 4. These little braces ($\langle . \rangle$) here mean you have taken the vector L_u and the vector R_v , did an element-wise product, and subsequently summed them up.

Okay, so what that represents is that you are taking a row of a big matrix L which is the vector L_u . This means that our user vector L_u is a row of this big matrix L . Remember that L_u represents how much a user likes different genres like action, romance, and so on. Similarly the movie vector, R_v is a column of

the big matrix R . Note that R_v is indexed over the same set of genres, or topics, for the movie, and has the same set of entries. Then in this matrix notations, according to the laws of matrix multiplication, if you multiply these two matrices L and R you will get the u - v -th entry of the ratings matrix. If you are not familiar with matrix multiplication that's okay. Just remember that this representation is a very compact way to store all of the vectors for all of the users and all of the movies. In this notation, the user vectors are column-wise stacked in the matrix R and the movie vectors are row-wise stacked in the matrix L . By multiplying these two matrices you will end up with an entire matrix which is same as the one I was writing before i.e. the users by movies matrix. So each individual entry of the ratings matrix is a combination of a specific row of the matrix L and a specific column of the matrix R shown in Figure 4.

Section 4: Discovering hidden structure by Matrix Factorization

Till this point in the discussion on matrix factorization, I have assumed that I know all the user topic vectors L_u which I stacked one above another to obtain the L matrix. Moreover, I assumed that I know all about the movie topic factors R_v which I placed side by side to obtain the R matrix. I said that by multiplying the two matrices L and R you will get this big prediction of ratings matrix that every user would give to every movie.

But the important thing to understand here is the fact that in real life you do not actually have this information. You do not have these features about the users, or about the movies. So instead, what I am going to do is that I am going to flip this problem on its head. And I am going to try and estimate the matrices L and R which is equivalent to estimating these topic vectors, or feature vectors, for every user and every movie based on our observed ratings. So these matrices, or these collections of topic vectors, are the parameters of our model. And like any Machine Learning technique, you are going to estimate these parameters from the available data. Now, can you guess what the data is in this case? Yes, the data is the observed ratings or the ratings already provided by users for the movies they saw. Remember that these were the white squares in our earlier matrix. I would like to reiterate that the users and the movies topic vectors are the parameters of this model. So your job is to estimate each of these vectors from our observed ratings. In other words, by only using these white cells, you have to estimate the L and R matrices. The question is - How are you going to do this? One technique typically used is to minimize the residual sum of squares (RSS). Let me explain how that is done. I said in the previous chapter that the predicted rating given by a user u to a movie v is given by $\langle L_u, R_v \rangle$. This is

obtained by taking an element-wise product of L_u and R_v and subsequently summing them up. Furthermore, the actual observed rating was denoted as $Rating(u, v)$. Now what you should do is that you should look at the difference between the observed rating and the predicting rating with the parameters L_u and R_v , and then you should square them. I will define this term as the residual sum of squares for the user vector L_u and the movie vector R_v . Next, to find the residual sum of squares of the parameter matrices L and R you have to find this term for all users and all movies in the data and add them. So you are going to include all u, v pairs where the ratings are available. And where are these available? They are the white squares in our Rating matrix as shown in Fig. 2. Okay, so what you are doing is that you are taking a given L matrix and R matrix and looking at your predictions. What you are interested in is in evaluating how well you did on all these white squares. So you are trying to find how well the L and R that you are using fit your observed ratings.

$$RSS(L_u, R_v) = (Rating(u, v) - \langle L_u, R_v \rangle)^2$$

$$RSS(L, R) = \sum_{\text{for all } u \text{ and } v} (Rating(u, v) - \langle L_u, R_v \rangle)^2$$

Then, when you want to estimate the parameter matrices L and R you search over all the user topic vectors and all the movie topic vectors, and find the combination of this huge space of parameters that best fits your observed ratings. Note that this method is called a matrix factorization model, because you are taking this huge Rating matrix, and approximating it with this factorization here. That is you are dividing the Rating matrix into two factors that are L and R as shown in Fig. 5. The key point you should keep in mind is that the output of this model is a set of estimated parameters here. In this notion, I will denote the estimated vectors L_u and R_v as \widehat{L}_u and \widehat{R}_v .

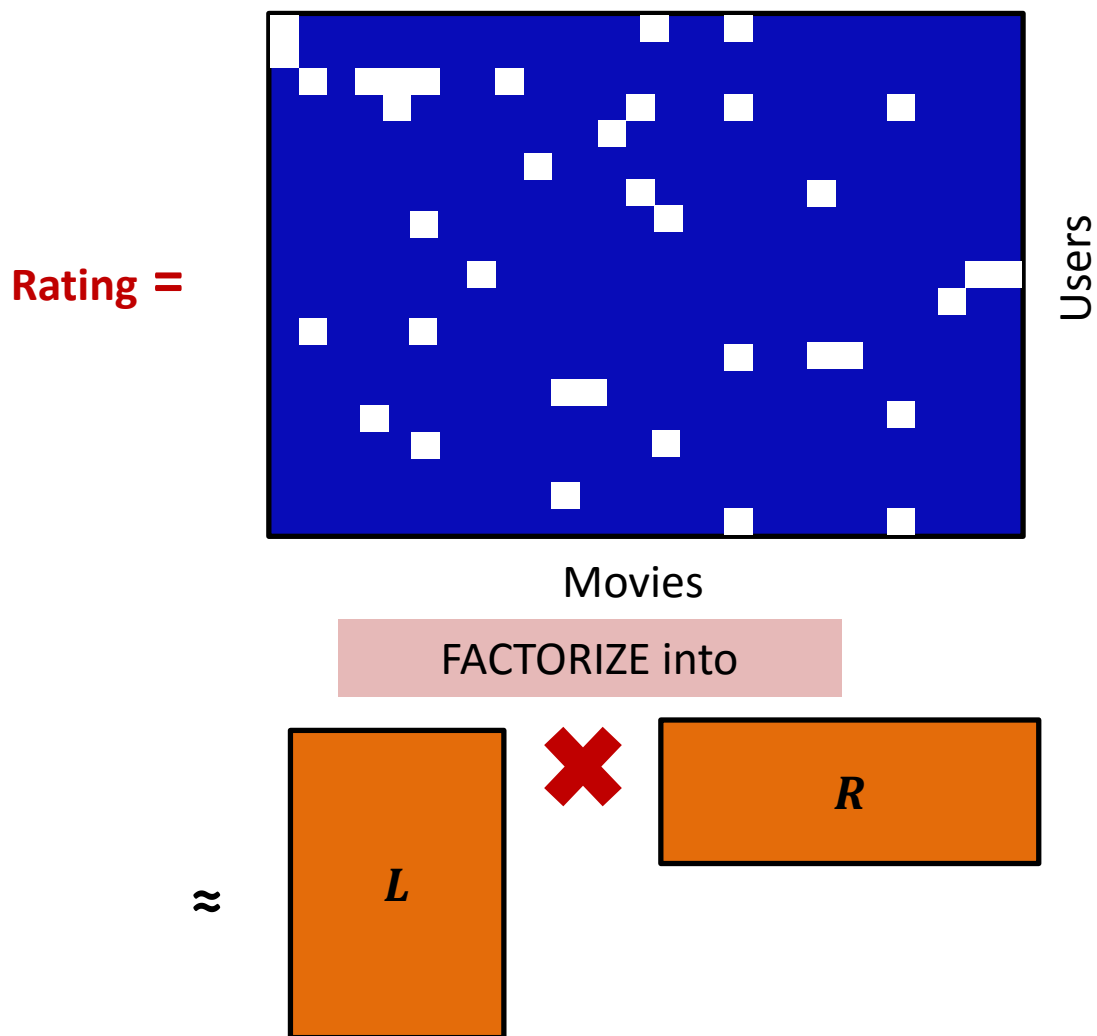


Figure 5: The Rating matrix composed of the known and unknown matrices are factorized into the L and R matrix.

There are a lot of very efficient algorithms from Linear algebra such as stochastic gradient descent that can be used for doing this factorization. Here, I shall discuss the algorithm termed as Stochastic Gradient Descent. Stochastic Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent. In Machine Learning, first an objective function is constructed by analyzing the problem at hand. Next, it's optima is found by applying Stochastic Gradient descent .

This section was about giving you an idea about how the movie recommendation model works. Consider that you have used a very efficient algorithms and obtained estimates of L and R . How do you form the predictions then? How do you fill in all these blue squares which was your goal from the beginning? Well, you just estimated \widehat{L}_u and \widehat{R}_v and you are going to form your predictions just as I described when you assumed that you actually knew these vectors. Please refer to Section 2 of this chapter if you need a refresher.

To conclude, matrix factorization is a very powerful tool and it's been proven useful on lots of different applications. But there's one limitation, and that's a problem that I talked about in the product recommendation case as well. It suffers from the cold-start problem i.e. this model still can't handle the inclusion of a new user or a new movie. So that's the case where you have no ratings either for a specific user or a specific movie. So that might be a new movie that arrives or a new user who arrives. This is a really important problem and one that Netflix faces all the time. How do you make predictions for these users or movies?

Section 5: Blended models to the rescue

Matrix Factorization is a very powerful method which suffers from the **cold start problem**. In other words, this approach cannot inherently handle the arrival of new users or new items. The reason behind this is that in Matrix factorization the features of users and items are learned by the model itself. Since nothing is known about the new user or item this method is unable to learn the corresponding vectors. So, the primary strength of this model i.e. its capability to learn features instead of handpicking them, gives rise to its main limitation.

To solve this issue you are going to combine different models. Remember that while discussing the feature-based classification approach I said that you have to handpick the features. You can handpick features that may capture things such as context, time of the day, user information, what I just saw, etc. On the other hand, Matrix Factorization discovers features that can capture groups of users who behave similarly. Now, the question is - how are you going to combine these two models to solve the cold start problem? Well, it is very straight-forward. When a new user comes who does not have any past purchase or ratings history, you should just use features specified by that user for example, the person's age, gender, etc. to recommend products or movies to him. In future, when you receive more and more

data about that user you can weigh more heavily on the Matrix Factorization approach and use the learned features more strongly while making recommendations.

Here, you have combined the ideas of a user-specified feature-based model with the learned features from matrix factorization. This helps you in reaping the benefits of both worlds. This idea of blending different models is very popular in Machine Learning. The Machine Learning term for these techniques is ensemble methods. The prime example of where this ensemble approach has been shown to have impact in recommenders systems was the classic example of the Netflix competition that we mentioned at the beginning of this course and shown in Fig. 6. This was a competition for who could provide the best predicted ratings for users on Netflix, and the data set consisted of a hundred million different user ratings and movies. There were almost 18,000 different movies, nearly 500,000 unique users, and the goal was to predict 3,000,000 ratings to the highest accuracy. The prize money was a whopping 1 million USD. This interested a lot of groups to take part in the competition and what I marked in this table is the winning team. The model they proposed blended over a hundred different models to get the performance shown. Hence, this notion of combining models to get performance that exceeds the performance of any of the individual models is a very common and very powerful technique employed in Machine Learning and that helped us too in solving the cold-start problem.

Netflix Prize				
COMPLETED				
Home Rules Leaderboard Update				
Leaderboard				
Showing Test Score. Click here to show quiz score				
Display top 20 leaders.				
Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries!	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

Figure 6: The list shows the top performers in the Netflix competition.

Chapter 5: Performance metrics for recommender systems

In the previous few chapters, I talked at great length about how to form predictions using different types of recommender systems. But a question is, how do you assess the difference in performance for these various systems you might consider using? Well imagine that you want to recommend kitchen items to a user, and Fig. 1 shows the set of all possible products that you can recommend. And the user likes a subset of these products which are represented by these blue squares. And our goal, of course, is to discover the products that the user likes from the purchases that they've made. In real life you don't actually know what products they like and that is what you want to discover.



Figure 1: Consider that this is the set of all kitchen items. The ones I like are marked by blue squares.

Section 1: A performance metric for recommender systems

While talking about measuring the performance of a Machine Learning system, people typically talk about classification accuracy. Classification accuracy is defined as the 100 multiplied by the ratio of the number of items classified correctly and the total number of items. So although it is a widely used

metric, it can't be used directly to measure the performance of a recommender system. This is because the classification accuracy metric works well when different classes of the data are balanced in terms of quantity. However, in the case of product recommendation this does not happen. Here, the data is divided into two classes – items which a user likes and items which the same user does not like. There are lots and lots of products out there, but typically a user is going to like a very small subset of them. So, the number of products that a user likes is much less than the number of products he/she doesn't like and hence the classes are imbalanced. So if you use classification accuracy, you can get a very good accuracy by just saying that the user won't like any of the items. So not recommending anything will provide a pretty good performance according to this metric.

Thus instead of using classification accuracy, you will be using a different metric or rather different metrics which are called **precision** and **recall**. Let's start by discussing how to calculate recall. Imagine again that you have designed a recommender system whose job is to recommend kitchen products to me. Figure 2 shows the set of kitchen products and the ones I like are marked by these white boxes. Suppose that out of all the products, the recommender system recommended these items i.e. the items marked by orange circles to me. To measure recall you have to first calculate all the items I like. That means all the products marked by blue boxes. Then, you have to find out how many of the items that I like were actually recommended to me i.e. all the products marked by both blue squares and orange circles. Then you have to take their ratio to calculate recall. Let us do that for this example. So the number of blue boxes is 5 and out of these 5 blue boxes three items were recommended to me. This means that 3 out of the 5 items I like were recommended to me. Hence, the recall here is three-fifth. Thus recall measures how much a recommended set of items cover the things that I'm interested in i.e. things that I actually like.



Figure 2: This figure shows the set of all kitchen items. The ones I like are marked by blue squares and the ones that were recommended to me were marked by orange circles.

$$\text{Recall} = \frac{\# \text{ liked and shown}}{\# \text{ liked}}$$

On the other hand, there is another metric called precision. While calculating precision what you are going to look at is all of the recommended items. Remember that when you were computing recall your world was the items I like i.e. the blue square boxes. That was the world that you looked at and everything else could disappear from the slide. But when you are computing precision, you have to look at all the recommended items i.e. the items that are highlighted by these orange circles. What you are trying to find is that what fraction of the recommended items do I actually like? Okay, so in this case there are three items that I like of the items shown to me. Remember that these are the products marked both by a green square and orange circle. And I was shown a total of 6 items i.e. the items

marked by orange circles and hence precision is given by 3/6. So this is how you calculate precision. And when you are thinking about precision you are basically trying to predict how much garbage I as a user have to look at compared to the number of items that I like. So, it measures that when I have a limited attention span, how much effort I am going to waste on the products that I do not like.

$$\text{Precision} = \frac{\# \text{ liked and shown}}{\# \text{ shown}}$$

Section 2: Optimal recommenders

In the previous section, I showed you two metrics to evaluate the performance of recommender systems. These are recall and precision. The goal of any recommender system is to maximize both of them. In this video, I will first discuss how you can maximize recall. Remember that recall measures how many of my liked items were actually recommended to me. And as the title of this slide suggest, there is a very easy way by which you can maximize recall. Just recommend everything. If you recommend everything, you're guaranteed to recommend the products that I like. So in this case what would recall be? Green squares represent the products I like and orange circles mark the products that you recommended. The value of recall would just be 1, because all 5 of the products I like were recommended. This happens since 5 out of the 5 products I like were shown to me.



Figure 2: In this figure, all of the available products have been recommended.

But what's the resulting precision of doing this? Well, it can actually be arbitrarily small. Since there are tons and tons of products out there and you have recommended all of them to me the denominator is huge. On the other hand, in this case, the number of products I actually like is a very small fraction of the total number of items shown to me and so the numerator is small. Thus precision becomes extremely low. So the strategy we used to maximize recall is not a great one.

Now think about what would be an optimal recommender? What's the best recommender you can imagine? Well, the best recommender is the one that recommends all the products I like but only the products I like. So everything that I do not like was never shown by the recommender system. This would be great since there is no wasted effort. The system is going to capture everything I like and you are going to make a lot of money with this recommender system. Think about what will be the values of precision and recall in this case? Both would be 1. And you figure out why? You can go through and verify that using the equations from the previous slides.

Chapter 6: Conclusion

Hi there! Congratulations on completing this book. Now, you know how recommender systems work and you are well equipped to build your own recommendation system. Let me reiterate what you learned in this book.

You started by looking into areas where recommender systems are typically used. For example, Amazon uses them for product recommendations, Netflix uses them for movie recommendations, Pandora uses them for song recommendations, Facebook and LinkedIn for friend recommendations and so on.

Next, you started learning about types of recommender systems and how to build them. First, you looked into popularity based systems that are used by news websites such as New York Times. Second, you learned how classifier approach can be used to implement recommender systems. Third, you looked into Collaborative Filtering and Co-occurrence matrix. You learned how to use this technique for building a product recommendation system like Amazon. Fourth, you learned about Matrix Factorization and understood how to create a movie recommendation system by using this method.

While learning about these systems, you looked into the pros and cons of each of these methods. Moreover, you learned how to evaluate the performance of a recommender system you have designed. I have collated all the information I shared in this course in my e-book. Moreover, you will find additional information in the book.

The knowledge you have gathered by taking this course will help you in designing a system perfectly suited for the task at hand.

Let me conclude by saying that it was an honor teaching you this subject. My motto is to teach Machine Learning in a simple way such that it does not seem difficult and becomes accessible to everyone. I believe that behind each successful Machine Learning algorithm there is a physical significance or intuition that makes it work. Mathematics is required just to validate it. And in this course I wanted to demonstrate these intuitions to you for the case of recommender systems. This book is my first attempt in doing so and if you have liked it please provide me with a good review. In near future, I will create more Machine Learning courses that serve this motto.