RECOMMENDER SYSTEMS

Project Report

Project Report submitted in partial fulfilment of the
requirement for the degree of

Bachelor of Technology in Information Technology

Under the supervision of

Dr. Vivek Sehgal

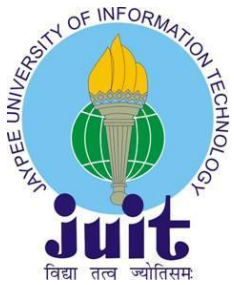
Associate Professor, Dept of Information Technology

By

Prakhar Anand Srivastava | 121427

Yash Mittal | 121430

to



Department Of Computer Science Engineering & Information Technology
Jaypee University of Information Technology, Waknaghat, Solan – 173234, Himachal Pradesh, INDIA

Certificate

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Recommender Systems"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat, is an authentic record of my own work carried out over a period from July 2015 to May 2016 under the supervision of **Dr Vivek Sehgal** (Associate Professor, Department of Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature) (Student Signature)

Yash Mittal, 121430 Prakhar Anand Srivastava, 121427

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)

Dr Vivek Sehgal

Associate Professor

Computer Science Department

Dated: 30/05/2016

Acknowledgement

We would like to thank everyone that has contributed to the development of this project, which is the final chapter of our Bachelor education in Information Technology at Jaypee University of Information Technology, Waknaghat, Solan.

We thank our supervisor Dr. Vivek Sehgal, for his guidance and valuable advice during this ongoing development of the project. We would also like to thank our parents who provided us with the opportunity to study in this university and enlightened our life and career.

Yash Mittal, 121430

Prakhar Anand Srivastava, 121427

(Information Technology)

Table of Content

List of Figures

Abstract

Recommender systems are a hot topic in this age of immense data and web marketing. Shopping online is ubiquitous, but online stores, while eminently searchable, lack the same browsing options as the brick-and-mortar variety. Visiting a movie rental store in person, a customer can wander over to the science fiction section and casually look around without a particular author or title in mind. Online stores often offer a browsing option, and even allow browsing by genre, but often the number of options available is still overwhelming.

Commercial sites try to counteract this overload by showing special deals, new options, and staff favorites, but the best marketing angle would be to recommend items that the user is likely to enjoy or need. Unless online stores want to hire psychics, they need a new technology. The field of machine learning has an ever-growing field of research in recommender systems, which fits the bill.

"Recommender systems are systems that based on information about a user's past patterns and consumption patterns in general, recommend new items to the user."

The research in this scope has led to the development of many methods to get through the opinion of other people, the relevant items for a specific person. Most of these methods work around the idea of finding similarities in people's tastes, using Social Network platforms, such as Facebook and Twitter. The prediction for a specific person is then based on the opinion of the most similar user to the person present in the network. This procedure is known as *Collaborative Filtering*. The other approach is *Content-based Filtering*. But one approach isn't enough in today's time when internet access is easy, social network usage is high and there is a huge library of media content and inventory lists. A *Hybrid Recommender System* is our best bet to tackle the issue of suggestions.

The idea of this project is to analyze different algorithms devised for making predictions and develop a system for recommending media content to the user according to his/her taste.

CHAPTER 1 – INTRODUCTION

1.1Introduction

Recommender systems are such an integral part of our lives and how we experience the web, whether it is on a browser, mobile application or on the desktop. They have become so pervasive and ubiquitous that we do not even notice them anymore. Every scalable system makes use of a recommendation system. It's not just the apps or web sites where they are used – they are available in our automobile's dashboard, our smart watch, even in our smart home devices.

Why Recommender Systems?

Over the past 25 years, since the birth of the World Wide Web, the Internet has matured a lot and today we're in the third generation of the web (WWW 3.0+). As the web moved from an owner model to a public crowdsourcing model and allowed people to contribute freely, it witnessed an exponential rise in the amount of content available, which was a good thing. But this led to two major problems:

(i) *Aggregation*: The amount of information became so large that it got tough to manage it while still being able to run a web service that was reachable to all parts of the world. This problem was solved by building worldwide content delivery and distribution networks, aided by the rise of NoSQL Database systems and decreasing storage costs.

(ii) *Searching*: The second major problem was how to ensure that the information is within the reach of the user and that the user does not get lost in the vast data dumps available. This proved to be an even bigger problem than aggregation since the data troves are vast and each user brings along with him/her a unique perspective and thus a unique search pattern. We are still trying to solve this problem today and are far from achieving a perfect solution to it. This is where recommender systems come into play.

In a nutshell, a recommender system is a system that helps predict user response to a variety of options. Predicting what the user might pick up next is the essential aim of a recommender system. There is an extensive class of web applications that involve predicting the user's response to options. Such a facility is called a *recommendation/recommender system.*

As opposed to the previous approach of presenting user with the whole information library, recommender systems ease this task by reducing the amount of information available to the user and providing recommendations and predictions tailored to a user's behaviour or profile, thereby making search simpler. This is the reason why Google is perhaps the best and most powerful recommender system in existence today.

Recommender systems have changed the way people find information, media content, products and even other people. They study patterns of behaviour to know what someone will prefer from among a collection of things he has never experienced. The technology behind recommender systems has evolved over the past 20 years into a rich collection of tools that enable the practitioner or researcher to develop effective recommenders.

1.2 Problem Statement

The most widely used of recommender systems is seen in the field of media and content based applications since they are the most used category of applications. Moreover, the libraries for media content are huge and searching for content according to one's taste becomes very difficult.

Recommender systems for content based applications include those systems built for the following categories:-

(i) Music
(ii) Movies
(iii) Television Shows
(iv) Videos
(v) Games
(vi) Books
(vii) News
(viii) Articles

Our problem statement is to build a recommender system for movies which is able to suggest a number of movies to the user which he/she may wish to watch in the near future. This may/may not use the social profile of a person from a third party social network such as Facebook, Twitter or his/her ratings on a review site such as Flixster, Metacritic or Rotten Tomatoes.

1.3 Objectives

The objectives of the project are listed as follows:-

(i) Select a topic with real-world applications
(ii) Survey the existing literature and review the work done
(iii) Study and analyse real recommender systems in place today
(iv) Design a recommendation engine
(v) Develop the recommender system
(vi) Perform an analysis of the system
(vii) Create an appropriate interface for the user

This is a listing of the goals we intend to achieve by working on this project.

1.4 Methodology

Agile Development

Agile methodology is an alternative to traditional project management, typically used in software development. It helps teams respond to unpredictability through incremental, iterative work cadences, known as sprints. Agile methodologies are an alternative to waterfall, or traditional sequential development.

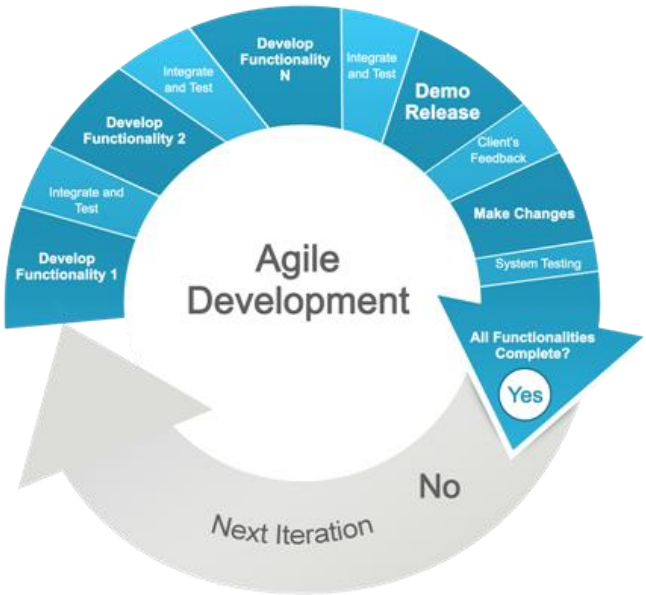


Figure 1 - Agile Methodology

1.5 Organization

The report follows the timeline in which the project work was done. It starts with an introduction to the topic – "Recommender Systems" which includes an abstract, a brief idea, objectives of the project and the methodology followed.

This is followed by a Literature Survey of the topic. It includes:-

(i) Applications of Recommender Systems
(ii) Types of Recommender Systems
(iii) Classification of Algorithms
(iv) Recommender Systems Studied

The report then follows up on the System Development of the project which includes analytical, computational, experimental, mathematical and statistical Model development.

This is followed by a Performance Analysis of the recommender system using various parameters and metrics. The report ends with a conclusion which includes Future Scope and Applications & Contributions.

CHAPTER 2 – LITERATURE SURVEY

2.1 Introduction

There is an extensive class of Web applications that involve predicting user responses to options. Such a facility is called a recommendation system. We shall begin this chapter with a survey of the most important examples of these systems. However, to bring the problem into focus, two good examples of recommendation systems are:

(i) Offering news articles to on-line newspaper readers, based on a prediction of reader interests.

(ii) Offering customers of an on-line retailer suggestions about what they might like to buy, based on their past history of purchases and/or product searches.

Recommendation systems use a number of different technologies. We can classify these systems into two broad groups.

- Content-based systems examine properties of the items recommended. For instance, if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the "cowboy" genre.
- Collaborative filtering systems recommend items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users.

2.2 Applications of Recommender Systems

We have mentioned several important applications of recommendation systems, but here we shall consolidate the list in a single place.

1. *Product Recommendations*: Perhaps the most important use of recommendation systems is at on-line retailers. We have noted how Amazon or similar on-line vendors strive to present each returning user with some suggestions of products that they might like to buy. These suggestions are not random, but are based on the purchasing decisions made by similar customers or on other techniques we shall discuss in this chapter.
2. *Movie Recommendations*: Netflix offers its customers recommendations of movies they might like. These recommendations are based on ratings provided by users, much like the ratings suggested in the example utility matrix of Fig. 9.1. The importance of predicting ratings accurately is so high, that Netflix offered a prize of one million dollars for the first algorithm that could beat its own recommendation system by 10%. The prize was finally won in 2009, by a team of researchers called "Bellkor's Pragmatic Chaos," after over three years of competition.

3. *News Articles*: News services have attempted to identify articles of interest to readers, based on the articles that they have read in the past. The similarity might be based on the similarity of important words in the documents, or on the articles that are read by people with similar reading tastes. The same principles apply to recommending blogs from among the millions of blogs available, videos on YouTube, or other sites where content is provided regularly.

2.3 Classification of Algorithms

2.3.1 Collaborative Filtering

The collaborative filtering is a technique for recommender systems that generates recommendations using the preferences and tastes given by others users of the system. This technique tries to simulate the collaboration in the real world between users that share opinions about recommendations and reviews.

In many cases, people have to choose between different alternatives without a complete knowledge of them. In these cases, the people believe in the recommendation of other familiar people or people whose opinion is valued by them.

The collaborative filtering systems use this idea, trying to get the users of the system that have the best opinion about an item for a user (based in his or her taste) and calculate the utility of the items for the specific user, using the opinion of the other users.

Appropriate Scenarios

Collaborative filtering can be applied in many domains, but for it to work efficiently, it is better to apply it in scenarios with some characteristics. The most important is that the evaluation of the items is based on subjective criteria (e.g. movies) or when the items have a lot of objective criteria and they need a subjective weight to choose between them (e.g. computers). In these situations, the collaborative filtering is very powerful, but if the recommendations are only based on objective criteria, the collaborative filtering does not make sense. It does not mean that the items have no objectives criteria to be evaluated. It means that the objective criteria are very similar, and they differ only in subjective criteria.

It is important too, that each user can find others users with similar tastes, because the rating of these similar users will be an important component of recommendations. The taste of the user cannot change a lot in short time, because then the previous ratings of this user do not represent his or her preferences and they become useless for predictions. And

about the data that is necessary in one scenario to use a Collaborative Filtering system, it is important that there are many ratings per item, to ensure a good inference of recommendation and predictions and that each user rates multiple items. The system needs enough information to provide recommendations with high quality.

Collaborative Filtering Algorithms

Most recommendation algorithms start by finding a set of customers whose purchased and rated items overlap the user's purchased and rated items. The algorithm aggregates items from these similar customers, eliminates items the user has already purchased or rated, and recommends the remaining items to the user. Two popular versions of these algorithms are *collaborative filtering* and *cluster models*. Other algorithms — including search-based methods — focus on finding similar items, not similar customers. For each of the user's purchased and rated items, the algorithm attempts to find similar items. It then aggregates the similar items and recommends them.

(i) Traditional Collaborative Filtering

A traditional collaborative filtering algorithm represents a customer as an *N*-dimensional vector of items, where *N* is the number of distinct catalog items. The components of the vector are positive for purchased or positively rated items and negative for negatively rated items. To compensate for best-selling items, the algorithm typically multiplies the vector components by the inverse frequency (the inverse of the number of customers who have purchased or rated the item), making less well-known items much more relevant. For almost all customers, this vector is extremely sparse. The algorithm generates recommendations based on a few customers who are most similar to the user. It can measure the similarity of two customers, A and B, in various ways; a common method is to measure the cosine of the angle between the two vectors:

$$similarity(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \bullet \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$

The algorithm can select recommendations from the similar customer's items using various methods as well; a common technique is to rank each item according to how many similar customers purchased it.

Using collaborative filtering to generate recommendations is computationally expensive. It is $O(MN)$ in the worst case, where M is the number of customers and N is the number of product catalog items, since it examines M customers and up to N items for each customer. However, because the average customer vector is extremely sparse, the algorithm's performance tends to be closer to $O(M + N)$. Scanning every customer is approximately $O(M)$, not $O(MN)$, because almost all customer vectors contain a small number of items, regardless of the size of the catalog. But there are a few customers who have purchased or rated a significant percentage of the catalog, requiring $O(N)$ processing time. Thus, the final performance of the algorithm is approximately $O(M + N)$. Even so, for very large data sets — such as 10 million or more customers and 1 million or more catalog items — the algorithm encounters severe performance and scaling issues.

It is possible to partially address these scaling issues by reducing the data size. We can reduce M by randomly sampling the customers or discarding customers with few purchases, and reduce N by discarding very popular or unpopular items. It is also possible to reduce the number of items examined by a small, constant factor by partitioning the item space based on product category or subject classification. Dimensionality reduction techniques such as clustering and principal component analysis can reduce M or N by a large factor.

Unfortunately, all these methods also reduce recommendation quality in several ways:

- If the algorithm examines only a small customer sample, the selected customers will be less similar to the user.
- Item-space partitioning restricts recommendations to a specific product or subject area.
- If the algorithm discards the most popular or unpopular items, they will never appear as recommendations, and customers who have purchased only those items will not get recommendations.

(ii) Cluster-based Collaborative Filtering

To find customers who are similar to the user, cluster models divide the customer base into many segments and treat the task as a classification problem. The algorithm's goal is to assign the user to the segment containing the most similar customers. It then uses the purchases and ratings of the customers in the segment to generate recommendations.

The segments typically are created using a clustering or other unsupervised learning algorithm, although some applications use manually determined segments. Using a similarity metric, a clustering algorithm groups the most similar customers together to form clusters or segments. Because optimal clustering over large data sets is impractical, most applications use various forms of greedy cluster generation. These algorithms typically start with an initial set of segments, which often contain one randomly selected customer each. They then repeatedly match customers to the existing segments, usually with some provision for creating new or merging existing segments. For very large data sets - especially those with high dimensionality - sampling or dimensionality reduction is also necessary.

Once the algorithm generates the segments, it computes the user's similarity to vectors that summarize each segment, then chooses the segment with the strongest similarity and classifies the user accordingly. Some algorithms classify users into multiple segments and describe the strength of each relationship. Classification can be done on the basis of Euclidean Distance:

$$dist(a,u) = \sqrt{\frac{\sum_{\{i \in S_a \cap S_u\}} (v_{ai} - v_{ui})^2}{|\{i \in S_a \cap S_u\}|}}$$

The rating will be the summation of ratings of the item by the users in the cluster divided by the number of users in the cluster:

$$\mu_{ki} = \frac{\sum_{\{u \in C_k | i \in S_u\}} v_{ui}}{|\{u \in C_k | i \in S_u\}|}$$

Cluster models have better online scalability & performance than collaborative filtering because they compare the user to a controlled number of segments rather than the entire customer base. The complex clustering computation is run offline.

However, recommendation quality is low. Cluster models group numerous customers together in a segment, match a user to a segment, and then consider all customers in the segment similar customers for the purpose of making recommendations. Because the similar customers that the cluster models find are not the most similar customers, the recommendations they produce are *less relevant*. It is possible to improve quality by using numerous fine grained segments, but then online user–segment classification becomes almost as expensive as finding similar customers using collaborative filtering.

Types of Collaborative Filtering

(i) User-based Collaborative Filtering

In this method, we predict the user behavior against a certain item using the weighted sum of deviations from mean ratings of users that previously rated this item and the user mean rate.

Also known as memory-based collaborative filtering, it is an effective technique and pretty easy to implement.

A weight is assigned to all users wrt similarity with the active user

→ The Rating value user 'u' gives to item 'i' is calculated as an aggregation of similar users' ratings

→ Find top-N users who are similar to user 'u', who also rated item 'i' positively, i.e., users who have maximum similarity with user 'u'

→ Compute a prediction from a weighted combination of the selected neighbours' ratings

(ii) Item-based Collaborative Filtering

E-commerce websites extensively uses recommendation algorithms to personalize its Web site to each customer's interests. Because existing recommendation algorithms cannot scale to tens of millions of customers and products, item-to-item collaborative filtering, scales to massive data sets and produces high-quality recommendations in real time.

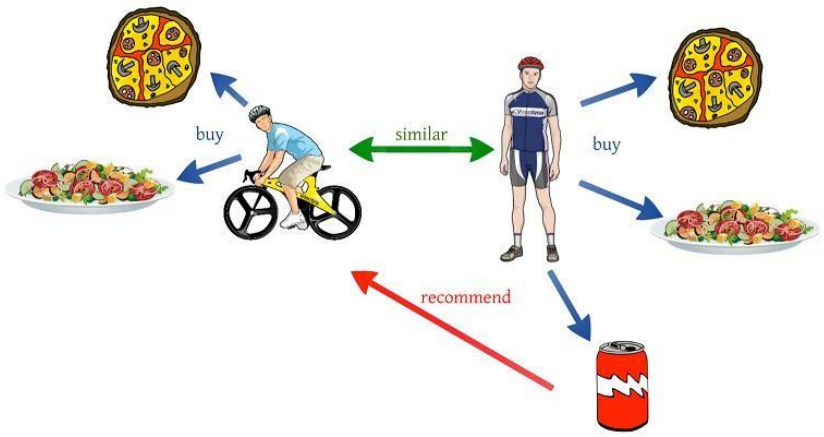


Figure 2 - User-based Collaborative Filtering

Rather than matching the user to similar customers, item-to-item collaborative filtering matches each of the user's purchased and rated items to similar items, then combines those similar items into a recommendation list.

Proposed first in 2003, it does NOT match similar users, but MATCHES *similar items*

→ Difference: Similar items bought vs Users who bought similar items

- Leads to faster online systems
- Results in improved recommendations
- *Pearson correlation* is the most common technique

Figure 3 - Item-based Collaborative Filtering

(iii) Model-based Approach

→ Develop models using data mining and machine learning algorithms to find patterns, based on a particular training dataset

→ Bayesian networks, clustering models, latent semantic models (Markov Decision process)

→ Parameter reduction can take place using Principal Component Analysis

→ Helps enterprises in user identification and classification, for targeted recommendation

→ Handles *sparsity* (gap in number of likes) better than memory-based ones

(iv) Hybrid Approach

→ Most successful approaches are a mix of memory-based and model-based approaches

→ Hard to recommend without a profile in times of social networks

→ Overcome sparsity (in user-based) as well as loss of information (in model-based)

→ Example: Google Newsstand recommender system

2.3.2 Content-based Filtering

The system learns to recommend items that are similar to the ones that the user liked in the past. The similarity of items is calculated based on the features associated with the compared items. In content-based recommendations, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes.

If the user has few purchases or ratings, content based recommendation algorithms scale and perform well. For users with thousands of purchases, however, it's impractical to base a query on all the items. The algorithm must use a subset or summary of the data, reducing quality. In all cases, ***recommendation quality is relatively poor***. The recommendations are often either too general (such as best-selling drama DVD titles) or too narrow (such as all books by the same author). Recommendations should help a customer find and discover new, relevant, and interesting items. Popular items by the same author or in the same subject category fail to achieve this goal.

Methodology

- Use an item-presentation algorithm (Ex: tf-idf)
- Create a user profile, by focusing on

→ A model of user's preference (gained explicitly or implicitly)

→ A history of user's interaction with the recommender system

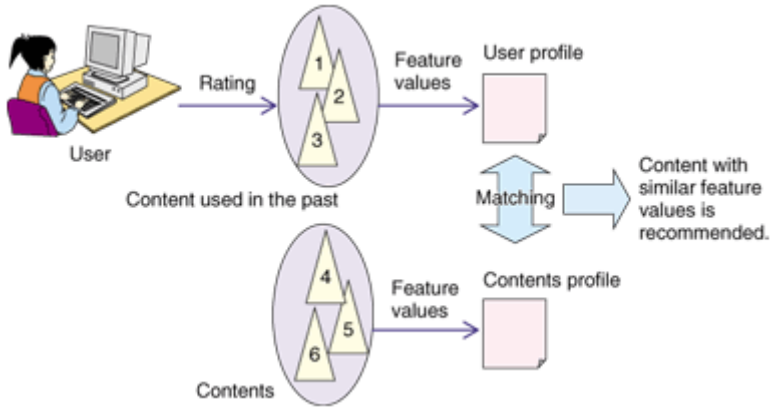


Figure 4 - Content-based Filtering

The main problems with the Content Based Filtering approach are:

a. *Domain and problem dependency*: For each application area one has to select the appropriate metadata describing the contents the best and ensure its availability. The availability of the right metadata content may not always be guaranteed, for instance, when the sites aggregate contents of different content providers, or products of numerous sellers/retailers. Typical examples are auction or classified sites.
b. *Scalability*: If the catalogue is large (millions of content items) then the selection of the right content requires comparing the user profile with all available content, which may take relatively long time.

2.3.3 Hybrid Recommender Systems

Implementation Techniques

→ Making content-based predictions & collaborative-based predictions separately and then combining

→ Adding content-based capabilities to a collaborative-based approach (& vice versa)

→ Unifying the approaches into one model

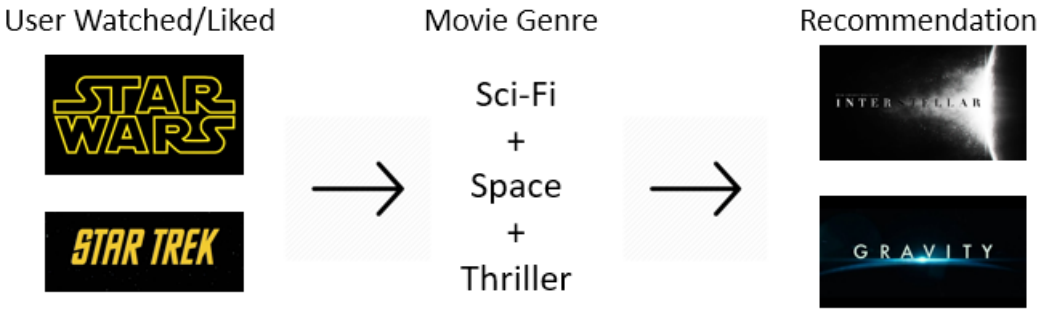


Figure 5 - Hybrid Recommender Systems

Example: Netflix

→ Collaborative filtering for comparing users' watching & searching habits

→ Content-based filtering for rating

Approaches may *combine*

(i) Collaborative

→ Information about rating profiles for different users

(ii) Content-based

→ Features associated with products and their ratings by users

(iii) Demographic-based

→ Recommend using ratings of users in a specific demographic dividend

(iv) Knowledge-based

→ Suggest products based on inferences about a user's needs & preferences

Hybridisation Techniques

- *Weighted* – Combine score of different recommendation components numerically
- *Switching* – System chooses among recommendation components & applies the selection
- *Mixed* – Recommendations from different recommenders are presented together
- *Feature Combination* – Combine features from multiple sources, give to a single algorithm
- *Feature Augmentation* – Use one recommendation technique to compute feature(s) set, and then provide to next technique
- *Cascade* – Assign priority to different recommenders
- *Meta-Level* – Use model of one recommendation technique as input to another.

2.4 Recommender Systems Studied

To get a good idea of how recommender systems work out in the real world, we studied a number of products, mainly web and mobile applications which made use of recommendations. Different types of applications have different needs for the kind of recommender system they use. We studied recommender systems being used in different types of online services.

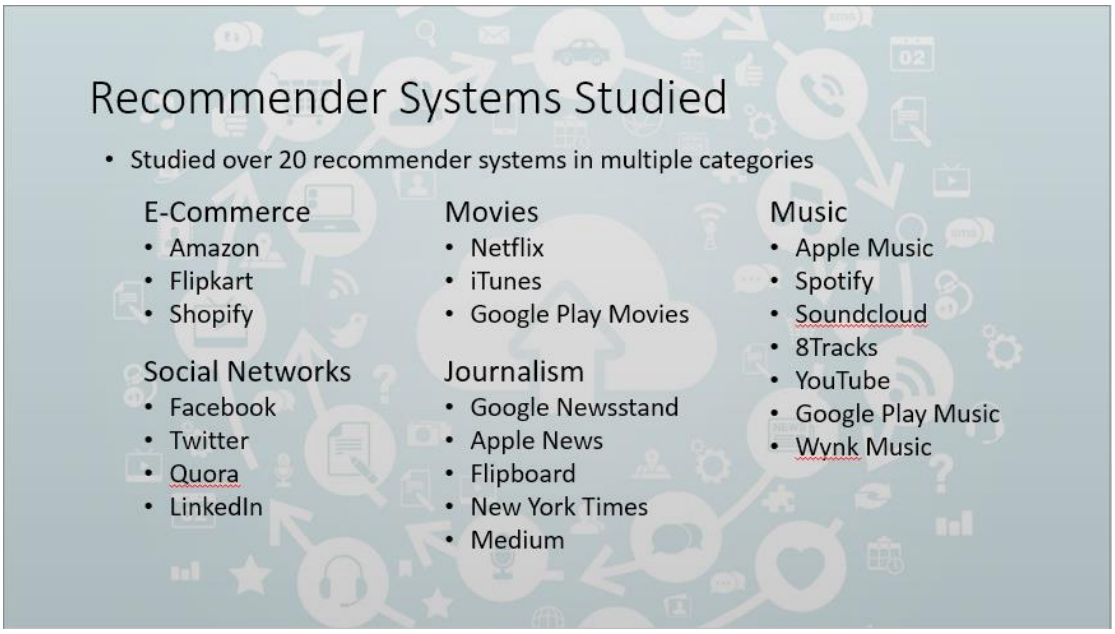


Figure 6 - Recommender Systems Studied

2.4.1. E-Commerce Recommendation Engines

Recommendation algorithms are best known for their use on e-commerce Websites, there they use input about a customer's interests to generate a list of recommended items. Many applications use only the items that customers purchase and explicitly rate to represent their interests, but they can also use other attributes, including items viewed, demographic data, subject interests, and favourite artists.

At Amazon.com, recommendation algorithms are used to personalize the online store for each customer. The store radically changes based on customer interests, showing programming titles to a software engineer and baby toys to a new mother. The click-through and conversion rates — two important measures of Web-based and email advertising effectiveness — vastly exceed those of untargeted content such as banner advertisements and top-seller lists.

E-commerce recommendation algorithms often operate in a challenging environment.

For example:

- A large retailer might have huge amounts of data, tens of millions of customers and millions of distinct catalogue items.
- Many applications require the results set to be returned in real-time, in no more than half a second, while still producing high-quality recommendations.
- New customers typically have extremely limited information, based on only a few purchases or product ratings.
- Older customers can have a glut of information, based on thousands of purchases and ratings.
- Customer data is volatile: Each interaction provides valuable customer data, and the algorithm must respond immediately to new information.

I. Amazon

Amazon.com uses recommendations as a targeted marketing tool in many email campaigns and on most of its Web sites' pages, including the high traffic Amazon.com homepage. Clicking on the "Your Recommendations" link leads customers to an area where they can filter their recommendations by product line and subject area, rate the recommended products, rate their previous purchases, and see why items are recommended (see Fig 1).

Figure 1. The "Your Recommendations" feature on the Amazon.com homepage. Using this feature, customers can sort recommendations and add their own product ratings.

Figure 7 - "Your Recommendations" Feature on Amazon.com

As Figure 2 shows, Amazon's shopping cart recommendations, which offer customers product suggestions based on the items in their shopping cart. The feature is similar to the impulse items in a supermarket checkout line, but our impulse items are targeted to each customer. Amazon.com extensively uses recommendation algorithms to personalize its Web site to each customer's interests. Because existing recommendation algorithms could not scale to Amazon's tens of millions of customers and products, Amazon developed its own. Amazon's algorithm, item-to-item collaborative filtering, scales to massive data sets and produces high-quality recommendations in real time.

Figure 2. Amazon.com shopping cart recommendations. The recommendations are based on the items in the customer's cart: The Pragmatic Programmer *and* Physics for Game Developers.

Figure 8 - Amazon.com Shopping Cart Recommendations

Rather than matching the user to similar customers, item-to-item collaborative filtering matches each of the user's purchased and rated items to similar items, then combines those similar items into a recommendation list.

To determine the most-similar match for a given item, the algorithm builds a similar-items table by finding items that customers tend to purchase together. We could build a product-to-product matrix by iterating through all item pairs and computing a similarity metric for each pair. However, many product pairs have no common customers, and thus the approach is inefficient in terms of processing time and memory usage.

The following iterative algorithm provides a better approach by calculating the similarity between a single product and all related products:

```
For each item in product catalog, I1
     For each customer C who purchased I1
          For each item I2 purchased by
               customer C
               Record that a customer purchased I1
               and I2
     For each item I2
          Compute the similarity between I1 and I2
```

Scalability

Amazon.com has more than 29 million customers and several million catalog items. Other major retailers have comparably large data sources. While all this data offers opportunity, it's also a curse, breaking the backs of algorithms designed for data sets three orders of magnitude smaller. Almost all existing algorithms were evaluated over small data sets. For example, the MovieLens data set contains 35,000 customers and 3,000 items, and the EachMovie data set3 contains 4,000 customers and 1,600 items. For very large data sets, a scalable recommendation algorithm must perform the most expensive calculations offline.

II. eBay

Feedback Profile**:** The Feedback Profile feature at eBay.com (www.ebay.com) allows both buyers and sellers to contribute to feedback profiles of other customers with whom they have done business. The feedback consists of a satisfaction rating (satisfied/neutral/dissatisfied) as well as a specific comment

about the other customer. Feedback is used to provide a recommender system for purchasers, who are able to view the profile of sellers. This profile consists of a table of the number of each rating in the past 7 days, past month, and past 6 months, as well as an overall summary (e.g., 867 positives from 776 unique customers). Upon further request, customers can browse the individual ratings and comments for the sellers.

III. Levi's

Style Finder: Style Finder allows customers of the Levi Straus (www.levis.com) website to receive recommendations on articles of Levi's clothing. Customers indicate whether they are male or female, then view three categories -- Music, Looks, Fun -- and rate a minimum of 4 "terms" or "sub-categories" within each. They do this by providing a rating on a 7-point scale ranging from "leave it" to "love it." They may also choose the rating of "no opinion." Once the minimum number of ratings are entered customers may select "get recommendations." Here, they are provided with thumbnails of 6 items of recommended clothing. Customers may provide feedback by use of the "tell us what you think feature" which allows them to enter an opinion rating for the recommended article of clothing. Feedback may change one or all of the six items recommended.

2.4.2. Social Networks - Recommendation Engines

There are findings in the sociological and psychological disciplines that point to the relevance of a person's social network in determining their tastes, preferences, and activities. The principle of homophily, for instance, is well established in the Social Networks field. McPherson et al. reported how "similarity breeds connection". They discovered that "people's personal networks are homogeneous with regard to many sociodemographic, behavioural, and intrapersonal characteristics". In other words, we share many attributes with the people close to us. Reversing this principle suggests that, if we have information about the connections in a person's network, we can infer some of the person's attributes.

It is possible that at least some of the similarities within a network are caused by the influence and interactions of the people in the network. People tend to remember information that was concretely given to them (that is, in personal interactions) better than abstract information (like statistical base rates). For example, Hogarth states that when considering to buy a certain car model we will likely give more thought to the direct advice of a friend than to each of the 100 respondents to a survey in a specialized magazine.

More specifically, Leskovec et al. discuss the phenomenon of information cascades, in which individuals adopt a new action or idea due to influence by others. In the most extreme cases, knowledge about a full network's behaviour determines the behaviour of its members –making a "top hits" list available in a music downloading website affects the popularity of the songs, and several different networks, kept in isolation of each other, prefer completely different songs, to the point that it is impossible to predict which will be the most popular songs for a network without observing the behaviour of the users in the network.

I. Facebook – News Feed

What is News Feed?

News Feed is the constantly updating list of stories in the middle of your home page. News Feed includes status updates, photos, videos, links, app activity and likes from people, Pages and groups that you follow on Facebook.

The order of stories in your News Feed is influenced by who posted the story, the number of comments and likes it received, and what kind of story it is (ex: photo, video, status update). This helps you to see the most interesting stories from the friends you interact with the most.

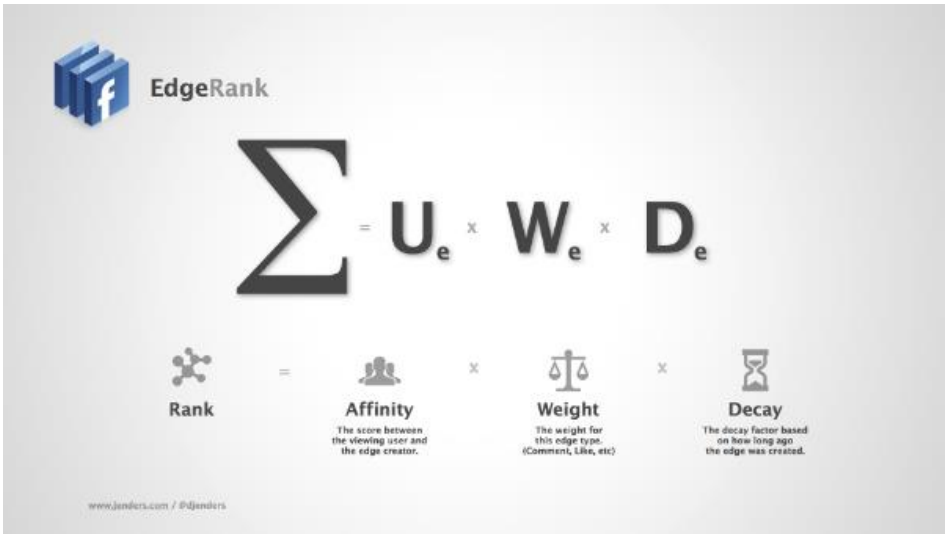


Figure 9 - Facebook EdgeRank Algorithm

How does News Feed work?

The stories that show in your News Feed are influenced by your connections and activity on Facebook. This helps you to see more stories that interest you from friends you interact with the most. The number of comments and likes a post receives and what kind of story it is (ex: photo, video, status update) can also make it more likely to appear in your News Feed.

So how does News Feed know which of those 1,500 stories to show? By letting people decide who and what to connect with, and by listening to feedback. When a user likes something, which tells News Feed that they want to see more of it; when they hide something, which tells News Feed to display less of that content in the future. This allows us to prioritize an average of 300 stories out of these 1,500 stories to show each day.

The News Feed algorithm responds to signals from you, including, for example:

- How often you interact with the friend, Page, or public figure (like an actor or journalist) who posted.
- The number of likes, shares and comments a post receives from the world at large and from your friends in particular.
- How much you have interacted with this type of post in the past.
- Whether or not you and other people across Facebook are hiding or reporting a given post.

Figure 10 - Facebook News Feed

In search of perfection, Facebook engineers constantly tweak the algorithm, using machine learning to study how people react and engage with content from friends and Pages. Two years ago, the algorithm was said to weigh as many as 100,000 factors — and getting more so over time. This is after EdgeRank was discontinued in 2013.

The feed must be completely personalized but still highly engaging to Facebook's users so they'll keep coming back and seeing more ads from the company's 2 million advertisers. But most users see only a sliver of the potential posts in their network each day. Facebook says the average user has access to about 1,500 posts per day but only looks at 300. (A user who scrolls endlessly will eventually see every post from their friends and a smattering of posts from Pages they follow.)

To ensure that those 300 posts are more interesting than all the rest, Facebook says it uses thousands of factors to determine what shows up in any individual user's feed. The

biggest influences are pretty obvious. How close you are to a person is an increasingly important metric, as judged by how often you like their posts, write on their Timeline, click through their photos or talk with them on Messenger, Facebook's chat service. The post-type is also a big factor, as Facebook hopes to show more links to people who click lots of links, more videos to people who watch lots of videos and so forth. The algorithm also assumes that content that has attracted a lot of engagement has wide appeal and will place it in more people's feeds.

But there are other, less intuitive factors to the algorithm. Use a phone with a slow mobile connection and you may see less video. Writing "congratulations" in a comment signals the post is probably about a big life event, so it will get a boost. Liking an article after you clicked it is a stronger positive signal than liking before, since it means you probably read the piece and enjoyed it.

The new team of human raters, which Facebook calls the "feed quality panel," are key to surfacing this meaningful content. Each day a typical panelist rates 60 stories that would actually appear in their own News Feeds on a 1 to 5 scale, judging how interesting they found the posts. They also reorder their own feeds to show how their priorities differ from the algorithm's, providing what Facebook calls a transposition score. And they write paragraph-long explanations for why they like or dislike certain posts, which are often reviewed in the News Feed engineers' weekly meetings. Facebook also regularly conducts one-off online surveys about News Feed satisfaction and brings in average users off the street to demo new features in its usability labs.

II. Twitter – Twitter Timeline

What's in your home timeline?

When you log in to Twitter, you'll land on your home timeline.

- A user's home timeline displays a stream of **Tweets** from accounts the user has chosen to follow on Twitter. New users may see suggested content powered by a variety of signals.
- One may see a summary of the most interesting Tweets he/she received since his/her last visit.
- A user may also see content such as promoted Tweets or Retweets in his/her timeline.
- Additionally, when Twitter identifies a Tweet, an account to follow, or other content that's popular or relevant, it may add it to the user's timeline. This means the user will sometimes see Tweets from accounts he/she doesn't follow. Twitter select each Tweet using a variety of signals, including how popular it is and how people in the user's network are interacting with it. Twitter's goal is to make the user's home timeline even more relevant and interesting.
- The newest updates are at the top. The user can reply, Retweet, or like a Tweet from within the timeline.
- Clicking anywhere on a Tweet in your timeline expands the Tweet, so the user can see photos, videos, and other information related to that Tweet.

Where the user might see other timelines:

- Timelines can also consist of collected Tweets from users in lists that you've curated or as search results.
- When you click on a list, you will see an aggregated stream of Tweets (a timeline) posted by the users included in that list.
- Similarly, when you perform a search, you'll see a timeline of Tweets that all match your search terms.

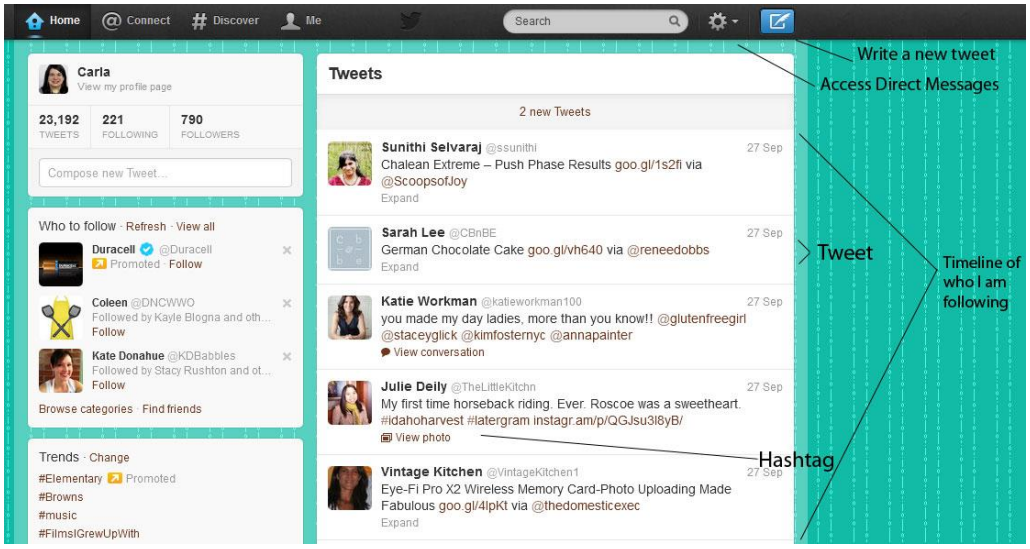


Figure 11 - Twitter Timeline

Tailored Suggestions:

Tailored suggestions make building a great timeline — filled with Tweets, links, media, and conversations from the people a user interested in — easier and faster. Twitter can make smarter and more relevant suggestions about who a user might enjoy following.

When a user first signs up for Twitter, it might offer him/her a unique list of relevant suggestions about who to follow so that the user can quickly experience the value and fun of Twitter. Twitter determines the people the user might enjoy following based on the user's visits to websites in the Twitter ecosystem (sites that have integrated Twitter buttons or widgets). Twitter might suggest people who are frequently followed by other Twitter users that visit the same websites.

If the person is a current user, this feature offers dynamic suggestions about people the user might enjoy following, keeping Twitter naturally aligned with the user's evolving interests. The user can find some of these suggestions in the Who to follow section of his/her Home timeline, as well as the Notifications and Me pages of the user's account.

2.4.3. Media Applications - Recommendation Engines

I. Netflix Recommendation Engine

Netflix has discovered through the years that there is tremendous value to its subscribers in incorporating recommendations to personalize as much of Netflix as possible. Personalization starts on its homepage, which consists of groups of videos arranged in horizontal rows. Each row has a title that conveys the intended meaningful connection between the videos in that group. Most of the personalization is based on the way we select rows, how we determine what items to include in them, and in what order to place those items.

Take as a first example the Top 10 row: this is Netflix's best guess at the ten titles you are most likely to enjoy. Of course, when Netflix says "you", it really means everyone in your household. It is important to keep in mind that Netflix' personalization is intended to handle a household that is likely to have different people with different tastes. That is why when it sees your Top10, you are likely to discover items for dad, mom, the kids, or the whole family. Even for a single person household Netflix wants to appeal to your range of interests and moods. To achieve this, in many parts of our system Netflix is not only optimizing for accuracy, but also for ***diversity***.

Figure 12 - Netflix - Diversity & Awareness of Recommendation Engine

Another important element in Netflix' personalization is ***awareness***. Netflix wants members to be aware of how Netflix is adapting to their tastes. This not only promotes trust in the system, but encourages members to give feedback that will result in better recommendations. A different way of promoting trust with the personalization component is to provide explanations as to why Netflix decides to recommend a given movie or show. Netflix is not recommending it because it suits its business needs, but

because it matches the information it has from you: your explicit taste preferences and ratings, your viewing history, or even your friends' recommendations.

On the topic of friends, in 2012, Netflix released its Facebook connect feature in 46 out of the 47 countries it operates in – all but the US because of concerns with the VPPA law. Knowing about your friends not only gives Netflix another signal to use in our personalization algorithms, but it also allows for different rows that rely mostly on your *social* circle to generate recommendations.

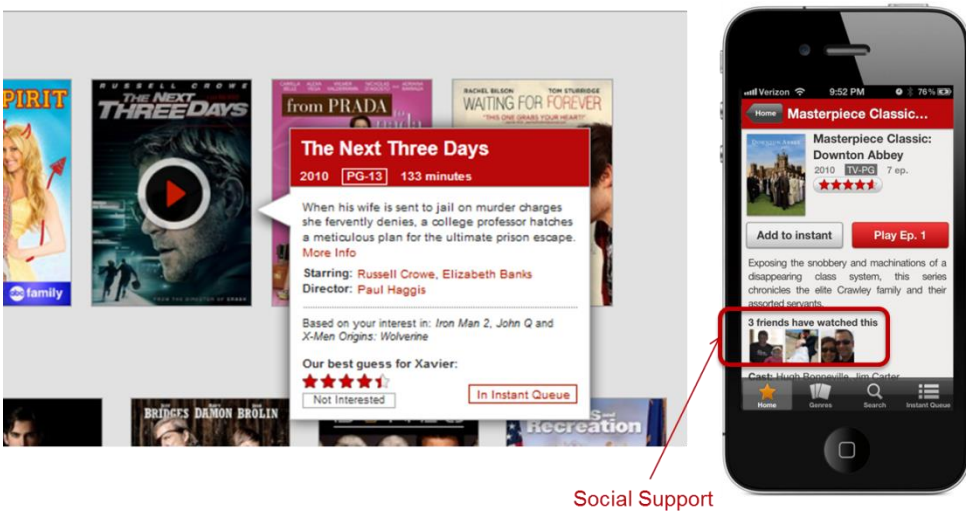


Figure 13 - Netflix - Social Connect

Some of the most recognizable personalization in our service is the collection of "*genre*" rows. These range from familiar high-level categories like "Comedies" and "Dramas" to highly tailored slices such as "Imaginative Time Travel Movies from the 1980s". Each row represents 3 layers of personalization: the choice of genre itself, the subset of titles selected within that genre, and the ranking of those titles. Members connect with these rows so well that we measure an increase in member retention by placing the most tailored rows higher on the page instead of lower. As with other personalization elements, *freshness* and diversity is taken into account when deciding what genres to show from the thousands possible.

Netflix presents an *explanation* for the choice of rows using a member's implicit genre preferences – recent plays, ratings, and other interactions --, or explicit feedback provided through our taste preferences survey. Netflix also invites members to focus a row with additional explicit preference feedback when this is lacking.

Similarity is also an important source of personalization in Netflix's service. Netflix thinks of similarity in a very broad sense; it can be between movies or between members, and can be in multiple dimensions such as metadata, ratings, or viewing data. Furthermore, these similarities can be blended and used as features in other models.

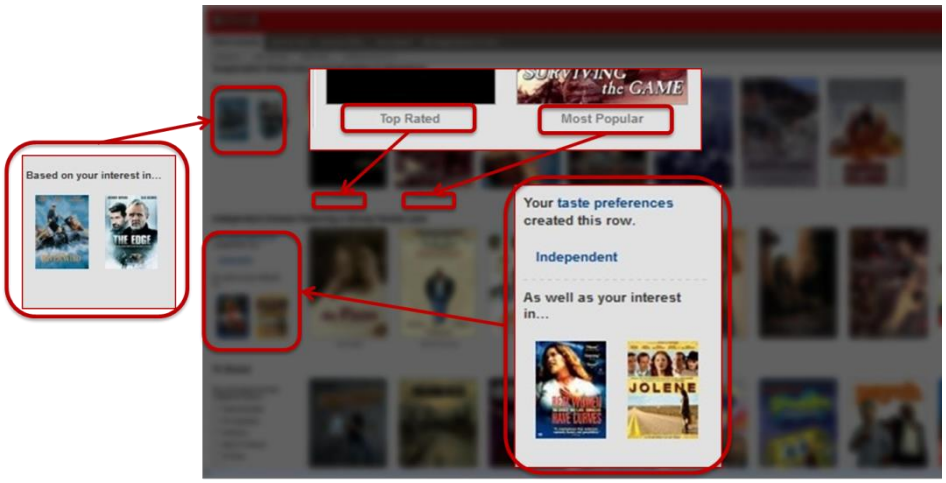


Figure 14 - Netflix - Explanation for Recommendations

Similarity is used in multiple contexts, for example in response to a member's action such as searching or adding a title to the queue. It is also used to generate rows of "adhoc genres" based on similarity to titles that a member has interacted with recently.

Ranking

The goal of recommender systems is to present a number of attractive items for a person to choose from. This is usually accomplished by selecting some items and sorting them in the order of expected enjoyment (or utility). Since the most common way of presenting recommended items is in some form of list, such as the various rows on Netflix, there needs to be an appropriate ranking model that can use a wide variety of

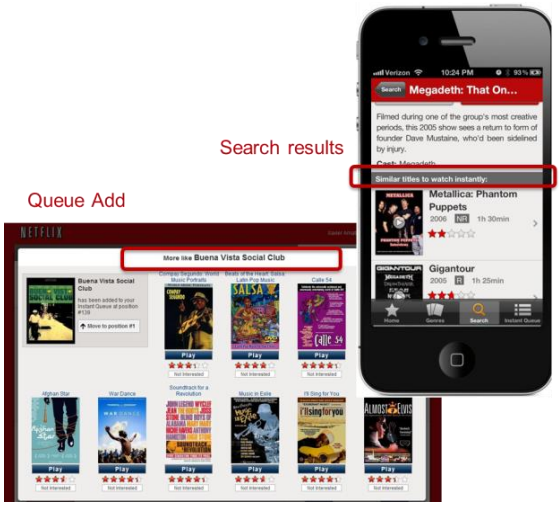


Figure 15 - Netflix - Similarity Results

information to come up with an optimal ranking of the items for each of Netflix's members. An obvious baseline is item popularity. The reason is clear: on average, a member is most likely to watch what most others are watching. However, popularity is the opposite of personalization: it will produce the same ordering of items for every member. Thus, the goal becomes to find a personalized ranking function that is better than item popularity, so we can better satisfy members with varying tastes.

There are many ways one could construct a ranking function ranging from simple scoring methods, to pairwise preferences, to optimization over the entire ranking. Netflix initially followed a very simple scoring approach - by choosing its ranking function to be a linear combination of popularity and predicted rating. This gives an equation of the form $f_{rank}(u,v) = w_1\, p(v) + w_2\, r(u,v) + b$, where u=user, v=video item, p=popularity and r=predicted rating. This equation defines a two-dimensional space like the one depicted below.

Once there is such a function, a set of videos can be passed through the function and be

Figure 16 - Netflix Ranking System

sorted in descending order according to the score. The question is - how can the weights w_1 and w_2 be set in the model (the bias b is constant and thus ends up not affecting the final ordering). In other words, in Netflix's simple two-dimensional model, how is it determined whether popularity is more or less important than predicted rating? There are at least two possible approaches to this. You could sample the space of possible weights and let the members decide what makes sense after many *A/B tests*. This procedure might be time consuming and not very cost effective. Another possible

answer involves formulating this as a *machine learning problem*: select positive and negative examples from your historical data and let a machine learning algorithm learn the weights that optimize your goal. This family of machine learning problems is known as *"Learning to rank"* and is central to application scenarios such as search engines or ad targeting.

Data & Models

Netflix has many relevant data sources and selects optimal algorithms to turn data into product features. Here are some of the data sources it uses to optimize user recommendations:

- Netflix has several billion item *ratings* from members and counting.
- Netflix mentions item *popularity* as a baseline. But, there are many ways to compute popularity. It can compute it over various time ranges, for instance hourly, daily, or weekly. Or, within a region or a group.
- Netflix receives several million *stream plays* each day, which include context such as duration, time of day and device type.
- Members add millions of items to their *queues* each day.
- Rich *metadata*: actors, director, genre, parental rating, and reviews.
- *Presentations*: How a recommendation affects user's actions. Netflix also observes the member's interactions with the recommendations: scrolls, mouse-overs, clicks, or the time spent on a given page.
- *Social data* - what connected friends have watched or rated.
- *Search terms* keyed in by the members in the Netflix service each day.
- *External data* such as box office performance or critic reviews. Other features such as demographics, location, language, or temporal data.

II. Spotify

Spotify is a service for streaming music. The catalogue of music can be accessed by downloading the Spotify client, an application developed to be small and very responsive to the user. Features include:

- An assortment of several million tracks
- An easy-to-use interface
- Streaming music from central servers, caching music on the client computers and replaying it between clients using Peer-to-Peer routing

Music streaming is a unique phenomenon in today's times because the user doesn't need to store any data and can stream any track online. Since most of our devices are connected 24x7 to the Internet, all the user needs to do is pick up a song by selecting an artist, song, or genre. The main question is – *what the user hears next*? And if the user will like the recommendations added to his infinite queue.

We have quite a few services which are unique:

- Spotify – scalable recommendation engine with good predictability
- Apple Music – recommendation engine + human curation
- Pandora – radio station service based on user's music like

Recommendations

While referring to recommendations, it corresponds to the normal usage of the term. Spotify is concerned with how much user *u* would like track *i* if he/she were to be presented with the track. Factors included in this is (among other) whether the track is good enough, whether there is an element of novelty, whether the track is different from other recommended tracks, and much more. Many of these aspects are hard to estimate, and in order to simplify the problem setting, we have to make some fundamental assumptions on what we are trying to achieve. To simplify the problem, Spotify treats predictions instead of recommendations. The predictions can be seen as probabilities estimated by the system, i.e., what is the probability that the user *u* will listen to track *i* next time? Instead of directly trying to find the best recommendations, Spotify tries to predict what the user will listen to in the future. The idea is that by predicting this, Spotify can present these results to the user immediately, and these tracks will hopefully be good and relevant recommendations. Whether predictions and recommendations are equivalent is another fundamental question. Treating predictions and recommendations enables us to analyse our methods, though there is many subtle differences.

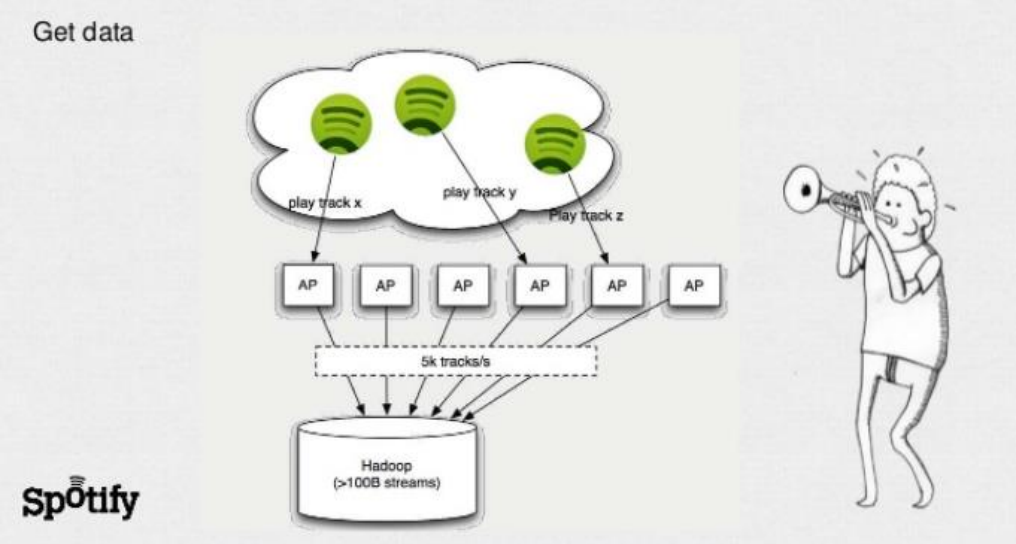


Figure 17 - How Spotify Gets Data

Representation

To treat the problem numerically, Spotify first defines a common way for the algorithms to treat predictions, in order to ensure consistency over all algorithms and provides a way to compare outputs from several algorithms. The time stamps of log entries are not taken into account. Spotify views a user's collected history of tracks in no particular order. Which last track the user listened to does not affect predictions more than any other tracks. Spotify also uses time dependency. It defines $P(N_{ui})$ as the probability that the next track the user u will listen to is track i. By definition, it obtains the normalization criterion

$$\sum_{i=0}^{n} P(N_{ui}) = 1$$

for all users u, i.e., the probability of any track being played as the next track is 1.0 (ignoring the very small possibility that the user has actually left Spotify and will never come back).

Methodology

Assuming predictions are good recommendations, recommending tracks to a user amounts to:

- Predict a set of tracks the user is most likely to play next.
- Recommending tracks which the user has already listened to will in general be less relevant, so remove all already played tracks.
- Recommend the top n remaining tracks to the user.

Having introduced the next track probability, the main objectives of the recommender system can be summarized as: Given a set of tracks that the user has listened to, together with the number of times:

- What is the probability that a user will listen to track *i* as the next track?
- Find the n tracks having the largest such probability

How to actually predict these probabilities will be discussed in the next section, where the models are introduced.

Models

(i) Collaborative Filtering

→ Item-oriented algorithms

→ User-oriented algorithms

(ii) Contextual or Feature-based Algorithms (Metadata Models)

→ Finding structures in whether users from specific countries do like songs in specific languages (for example, users from Germany are more likely to listen to German lyrics)

→ Finding correlations between music tastes and demographic data (such as age, sex, etc.)

→ Using editorial data on similar artists to produce recommendations (earlier system at Spotify)

→ Analysing audio contents of all tracks

Audio-based Algorithm

Audio-based methods are likely to be the best feature-based methods, having prediction accuracy on par with collaborative filtering methods. Using a feature extraction tool, one can produce a feature vector for each track.

Ideally, the feature vectors should express the human perception as much as possible. Common values that can be extracted are:

- Rhythm (most often measured in BPM, Beats Per Minute)
- Major/minor key
- Rhythm complexity (regularity/irregularity, etc.)
- *Timbre*, which is an umbrella term for many different aspects of audio

- *Mel-Frequency Cepstral Coefficients* (MFCC) are automatically extracted coefficients corresponding to timbre

The Ensemble Method

Instead of developing one algorithm to produce predictions, Spotify develops an array of them, the rationale being that different algorithms best handle different aspects of data and data scalability. These predictions can be combined to produce one final set of predictions. This method is commonly referred to as the ensemble method.

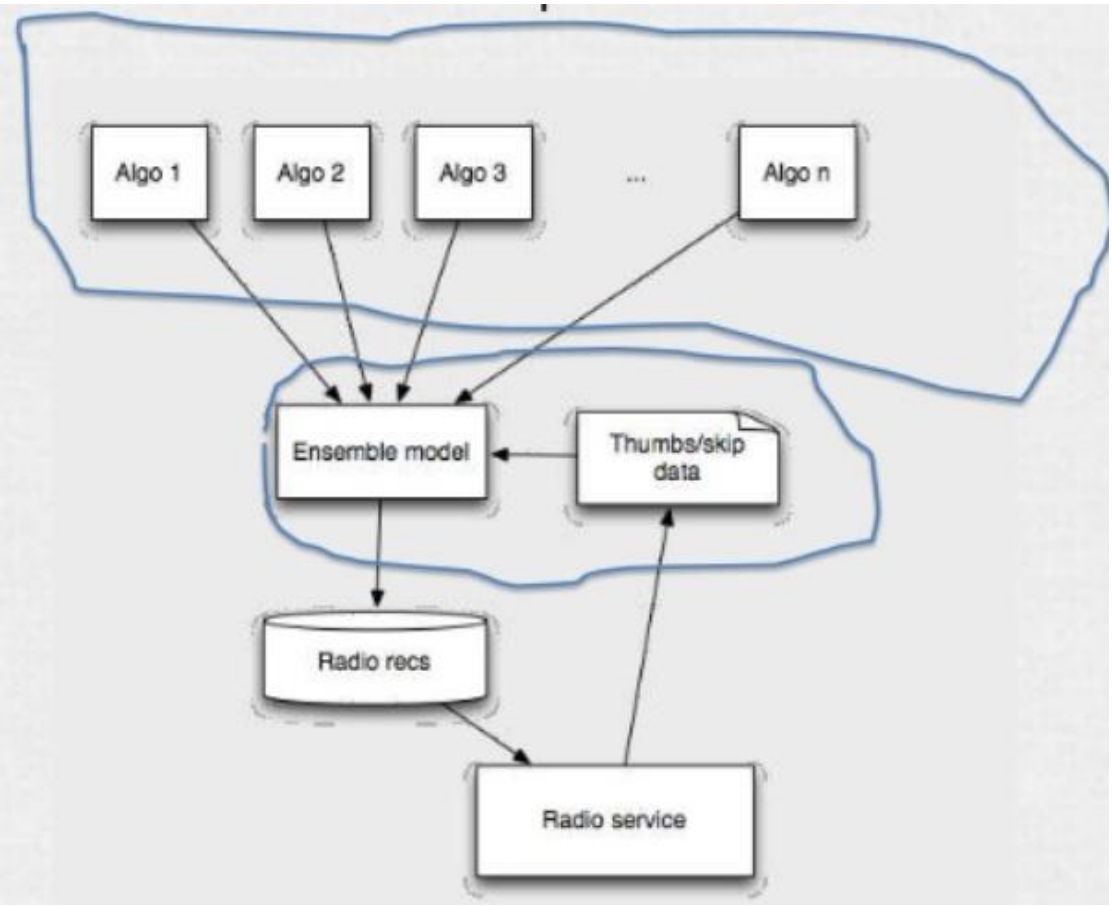


Figure 18 - Spotify's Ensemble Model

III. Apple Music

Apple Music offers a section called *For You* which provides a personalized selection of music based on your listening habits. According to Apple, "their experts handpick songs, artists, and albums based on what you listen to and like" while giving you "their take on the freshest and most relevant stuff around."

How "For You" Recommendations Work?

Apple's recommendation system on Apple Music is basically a supercharged Genius system that iTunes has been using for years now. Telling Apple about music you like helps them deliver better and better suggestions to you over time. The recommendation engine takes into account the following:

- *Hearts* – Hearting anything that's playing through Apple Music helps the system better tailor For You recommendations to your tastes. You can heart the following items on Apple Music by tapping the heart icon in Now Playing view:
  - → Any song in your personal library
  - → Any song available for streaming in Apple Music's catalogue
  - → Songs found through search
  - → Songs played from Beats 1 and Apple Music Radio
  - → Apple-curated playlists
- *Plays* – Apple Music's recommendation engine pays attention to what you actually play to help surface similar content you may find interesting. It's important to note that Apple Music's recommendation engine takes into account full plays, but discards skips.
- *Your library* – Songs you've downloaded from the iTunes Store, ripped from CDs or imported into iTunes from other sources are analyzed. Your personal library data, along with any music manually added from Apple Music to your library, influences music you get exposed to in the For You section.
- *Genres and bands you're into* – As part of Apple Music's setup procedure, Apple asks you to tell them which songs and genres you like. This data helps the system quickly learn what you're into.

These aforementioned items directly influence what content you'll be exposed to when browsing 'For You' recommendations.

The recommendation engine considers each and every like you make. “Whether you love a song or not, your feedback helps our suggestions get better and better,” by paying attention to what you actually play. Also, the cold start problem is solved in Apple Music by asking the user to select genres and artists at the initial stage of setup. This ensures that the system has at least something to offer the user, even if he/she hasn’t streamed any tracks yet.

Figure 19 - Apple Music - Initial Selection

CHAPTER 3 - SYSTEM DEVELOPMENT

The application segment developed and mentioned further is for generating recommendations for movies. For the development, the Movie-Lens dataset is being used. It consists of info for about 1,000,000 ratings (1-5) from 11000 users on 30,000 movies.

The recommendations will be made through the following procedures. First the data is analyzed and the dataset is trained for mining the rules for frequent itemsets. Association Rule Mining is then performed on the dataset to extract the association rules with length=3, conf=0.6, minsup=0.8. In data mining, association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. The Algorithm used for the association rule mining is Apriori Algorithm. In computer science and data mining, Apriori is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing).

The mined rules are then used to perform cluster analysis. K-means Nearest Neighbour Algorithm is used to find out the clusters from the data-set according to the rules mined earlier by using the Apriori Algorithm. It is most often used for classification, although it can also be used for estimation and prediction. K-Nearest Neighbour is an example of instance-based learning, in which the training data set is stored, so that a classification for a new unclassified record may be found simply by comparing it to the most similar records in the training set. The Cluster Analysis gives us the information about the movies with similar behaviour with respect to their ratings & genre.

3.1 Apriori Algorithm

The definition of association rules can be given as: Let I be a set of items. Each *transaction*, T, is also a set of items, called itemset, and $T \subseteq I$. The entire dataset, D, consists of uniquely identifiable transactions. An association rule is defined as an implication of the form $X \rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The sets of items (for short itemsets) X and Y are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively. Various metrics describe the utility of an association rule. The most common ones are the percent of all containing $A \cup B$, called the *support*, and the percent of transactions containing B among transactions containing A, called the confidence of the rule.

The Apriori Algorithm to mine association rule takes the dataset as input along with the minimum support and confidence thresholds, and its final output is the list of all association rules whose confidence and support are above the minimum thresholds. As an intermediate step, it produces frequent itemsets. An itemset is frequent if its support is greater than or equal to the minimum support specified by the user. An itemset of size k is called K-itemset.

Support:

The support supp(X) of an itemset X is defined as the proportion of transactions in the data set which contain the itemset.

supp(X)= no. of transactions which contain the itemset X / total no. of transactions

Confidence:

The confidence of a rule is defined: $conf(X \rightarrow Y) = supp(X \cup Y) / supp(X)$

The implementation of the Apriori Algorithm discussed above is done in R language in order to mine frequent itemsets from the dataset that has been considered for the project. The R script was executed using RStudio.

3.1.1 Implementation of Apriori in R

In R to study the association mining we use two important packages called *arules* and *arulesViz* written be Michael Hahsler. In order to implement the Apriori Algorithm in R, *arules* package of R is used. The *arules* is used for mining transaction records and extracting association rules. The package arulesViz is used as a presentation layer to visualize the association rules.

Package: arules

Version: 1.3-1

Description:

Mine frequent itemsets, association rules or association hyperedges using the Apriori algorithm. The Apriori algorithm employs level-wise search for frequent itemsets. The implementation of Apriori used includes some improvements (e.g., a prefix tree and item sorting).

Usage:

```
apriori(data, parameter = NULL, appearance = NULL, control = NULL
```

Arguments:

data

object of class transactions or any data structure which can be coerced into transactions (e.g., a binary matrix or data.frame).

parameter

object of class APparameter or named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 10.

appearance

object of class APappearance or named list. With this argument item appearance can be restricted (implements rule templates). By default all items can appear unrestricted.

control

object of class APcontrol or named list. Controls the algorithmic performance of the mining algorithm (item sorting, etc.)

Details:

Calls the C implementation of the Apriori algorithm by Christian Borgelt for mining frequent itemsets, rules or hyperedges.

Note: Apriori only creates rules with one item in the RHS (Consequent)!

Note: The default value in APparameter for minlen is 1. This means that rules with only one item (i.e., an empty antecedent/LHS) like {} => {beer} will be created. These rules mean that no matter what other items are involved the item in the RHS will appear with the probability given by the rule's confidence (which equals the support). If you want to avoid these rules then use the argument parameter=list(minlen=2)

Values:

Returns an object of class rules or itemsets.

Package: arulesViz

Description:

Even after filtering the rules using some constraint measures, we might end up with large list of interesting rules outputted by the aprior rule function. And it not viable to go through all rules one by one. We can use Visualization technique to get deep insight about the rules. We can use arulesViz package to visualize the association rules. The package arulesViz gives us ability to draw different charts and graphs without getting our hard much dirty in coding.

Scatter plot: Default plot for arulesViz package is scatter plot. In this plot association rules are plotted again axes. Usually X-axis corresponds to support and Y-axis corresponds to confidence. However these shading represents an additional coordinate that can be represented in this two dimensional space. Here color coordinate represent the lift value for each point on the scatter plot.

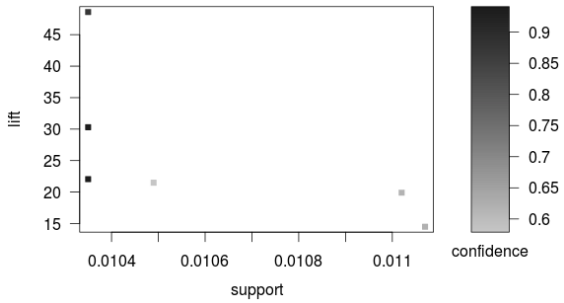


Figure 20 - Scatter Plot

plot(rules, measure=c("support", " confidence "), shading="confidence")

Console F:/JUIT/Sem - 7/Project/R & Stuff/Recommender Systems/

creating S4 object ... done [0.00s].

	lhs		rhs	support	confidence	lift
1	{}	=>	{movieId=A }	0.4	0.4000000	1.000000
2	{}	=>	{rating=FOUR}	0.5	0.5000000	1.000000
3	{}	=>	{userId=ONE}	0.9	0.9000000	1.000000
4	{movieId=A}	=>	{rating=FOUR}	0.1	1.0000000	2.000000
5	{movieId=A}	=>	{userId=ONE}	0.1	1.0000000	1.111111
6	{userId=TWO}	=>	{movieId=B}	0.1	1.0000000	10.000000
7	{movieId=B}	=>	{userId=TWO}	0.1	1.0000000	10.000000
8	{userId=TWO}	=>	{rating=TWO}	0.1	1.0000000	10.000000
9	{rating=TWO}	=>	{userId=TWO}	0.1	1.0000000	10.000000
10	{movieId=B}	=>	{rating=TWO}	0.1	1.0000000	10.000000
11	{rating=TWO}	=>	{movieId=B}	0.1	1.0000000	10.000000
12	{movieId=E}	=>	{rating=THREE}	0.2	1.0000000	5.000000
13	{rating=THREE}	=>	{movieId=E}	0.2	1.0000000	5.000000
14	{movieId=E}	=>	{userId=ONE}	0.2	1.0000000	1.111111
15	{rating=THREE}	=>	{userId=ONE}	0.2	1.0000000	1.111111
16	{movieId=D}	=>	{rating=FOURPF}	0.2	1.0000000	5.000000
17	{rating=FOURPF}	=>	{movieId=D}	0.2	1.0000000	5.000000
18	{movieId=D}	=>	{userId=ONE}	0.2	1.0000000	1.111111
19	{rating=FOURPF}	=>	{userId=ONE}	0.2	1.0000000	1.111111
20	{movieId=A }	=>	{rating=FOUR}	0.4	1.0000000	2.000000

Figure 21 - Top 20 Results of Apriori

3.2 K-Nearest Neighbour Algorithm

The k-nearest neighbour algorithm assigns the classification of the most similar record or records. But just how do we define *similar*?

For example, suppose that we have a new action movie with a rating of 4.5/5. Which movie is this more similar to, a comedy | action movie of 80's with a rating of 4 or a latest thriller movie with a rating 4. Data analysts define distance metrics to measure similarity. A distance metric or distance function is a real-valued function d, such that for any coordinates x, y, and z:

1. $d(x,y) \geq 0$, and $d(x,y) = 0$ if and only if $x = y$

2. $d(x,y) = d(y,x)$

3. $d(x,z) \leq d(x,y) + d(y,z)$

Statement (1) assures us that distance is always greater than or equal, and the only way for distance to be zero is when the coordinates (e.g., in the scatter plot) coincide.

Statement (2) state the commutativity property, considering an example, that the distance from New Delhi to Hyderabad is the same as the distance from Hyderabad to New Delhi.

Finally, Statement (3) indicates the triangle inequality, this can be understood as, introducing a third point can never shorten the distance between two other points.

The most common distance function is Euclidean distance, which represents the usual manner in which humans think of distance in the real world:

dEuclidean(x,y) = = $\sqrt{(\sum_i (xi - yi)^2)}$

Where x = x1, x2, . ., xm, and y = y1, y2, . . , ym represent the m attribute values of 2 records.

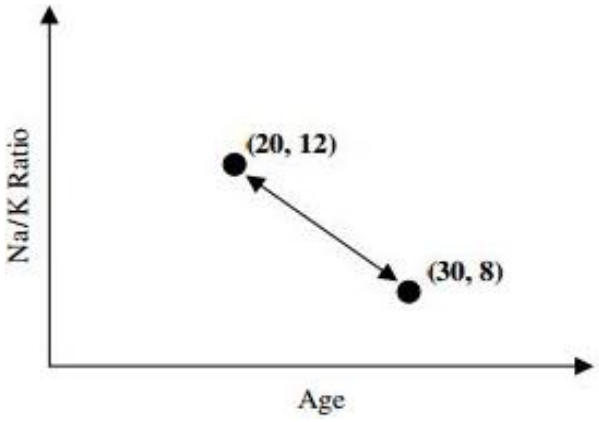


Figure 22 - Visualisation of Euclidean Distance

For example, suppose that patient A is x1 = 20 years old and has a Na/K ratio of x2 = 12, while patient B is y1 = 30 years old and has a Na/K ratio of y2 = 8. Then the Euclidean distance between these points, is shown as:

$$d_{\text{Euclidean}}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_i (x_i - y_i)^2} = \sqrt{(20-30)^2 + (12-8)^2}$$
$$= \sqrt{100+16} = 10.77$$

When measuring distance, however, certain attributes that have large values, such as income, can overwhelm the influence of other attributes which are measured on a smaller scale, such as years of service. To avoid this, one should make sure to normalize the attribute values.

For continuous variables, the min–max normalization or Z-score standardization, may be used:

Min–max normalization:

$$X^* - \frac{X - \min(X)}{\text{range(X)}} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Z-Score standardisation:

$$X^* = \frac{X - \text{mean}(X)}{\text{SD}(X)}$$

For categorical variables, the Euclidean distance metric is not appropriate. Instead, we may use Hamiltonian Distance metric, defining a function, “hamDist()” used to compare the ith attribute values of a pair of records, as follows:

$$\text{hamDist(xi, yi)} = \begin{cases} 0 & \text{if } x_i = y_i \\ 1 & \text{otherwise} \end{cases}$$

Where xi and yi are categorical values. We may then substitute different (xi, yi) for the ith term in the Euclidean distance metric above.

3.2.1 Implementation of K-Nearest Neighbour Algorithm

In R for cluster analysis, we use a package named *kknn*. This package is used for weighted k-Nearest Neighbors Classification, Regression and spectral Clustering The complete list of functions can be displayed with library(help = kknn). The package is written by Klaus Schliep and Klaus Hechenbichler and maintained by: Klaus Schliep.

Package: kknn

Version: 1.3.0

Description:

The package is titled as Weighted K-Nearest Neighbours. It has dependencies based on the version of R, i.e. version >= 2.10. It imports the following modules *igraph*(>=0.6), *Matrix*, *stats* and *graphics*

Method: kknn()

Description:

Performs k-nearest neighbour classification of a test set using a training set. For each row of the test set, the k nearest training set vectors are found, and the classification is done via the maximum of summed kernel densities. In addition even ordinal and continuous variables can be predicted.

Usage:

kknn(formula = formula(train), train, test, na.action = na.omit(), k = 7, distance = 2, kernel = "optimal", ykernel = NULL, scale=TRUE, contrasts = c('unordered' = "contr.dummy", ordered = "contr.ordinal")) kknn.dist(learn, valid, k = 10, distance = 2)

Arguments:

formula A formula object.

train Matrix or data frame of training set cases.

test Matrix or data frame of test set cases.

learn Matrix or data frame of training set cases.

valid Matrix or data frame of test set cases.

na.action A function which indicates what should happen when the data contain 'NA's. k Number of neighbours considered.

distance Parameter of Euclidean distance.

kernel Kernel to use. Possible choices are "rectangular" (which is standard unweighted knn), "triangular", "epanechnikov" (or beta(2,2)), "biweight" (or beta(3,3)), "triweight" (or beta(4,4)), "cos", "inv", "gaussian", "rank" and "optimal".

ykernel Window width of an y-kernel, especially for prediction of ordinal classes.

scale logical, scale variable to have equal sd.

contrasts A vector containing the ’unordered’ and ’ordered’ contrast

Details:

This nearest neighbour method expands knn in several directions. First it can be used not only for classification, but also for regression and ordinal classification. Second it uses kernel functions to weight the neighbours according to their distances. In fact, not only kernel functions but every monotonic decreasing function f(x)∀x > 0 will work fine. The number of neighbours used for the "optimal" kernel should be [(2(d + 4)/(d + 2))(d/(d + 4))k], where k is the number that would be used for unweighted knn classification, i.e. kernel="rectangular". This factor (2(d + 4)/(d + 2))(d/(d + 4)) is between 1.2 and 2

Value:

kknn returns a list-object of class kknn including the components

fitted.values Vector of predictions.

CL Matrix of classes of the k nearest neighbours.

W Matrix of weights of the k nearest neighbours.

D Matrix of distances of the k nearest neighbours.

C Matrix of indices of the k nearest neighbours.

prob Matrix of predicted class probabilities.

response Type of response variable, one of continuous, nominal or ordinal.

distance Parameter of distance.

call The matched call.

terms The ’terms’ object used.

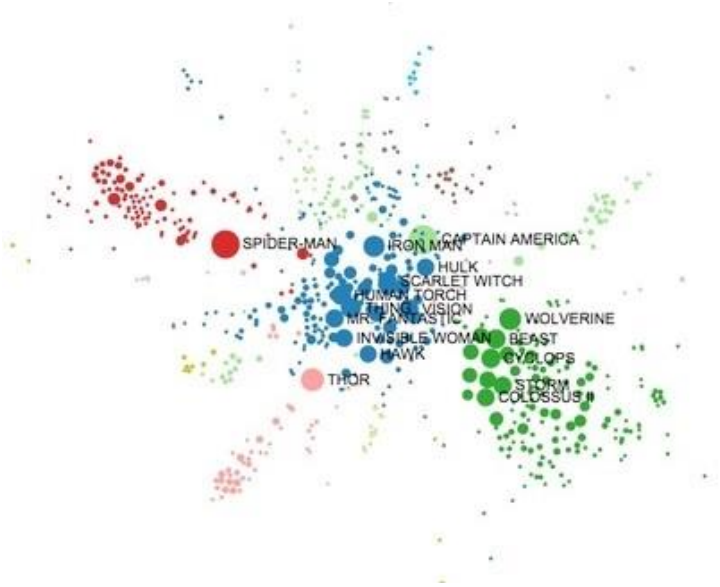


Figure 24 - Clustering Iteration 1

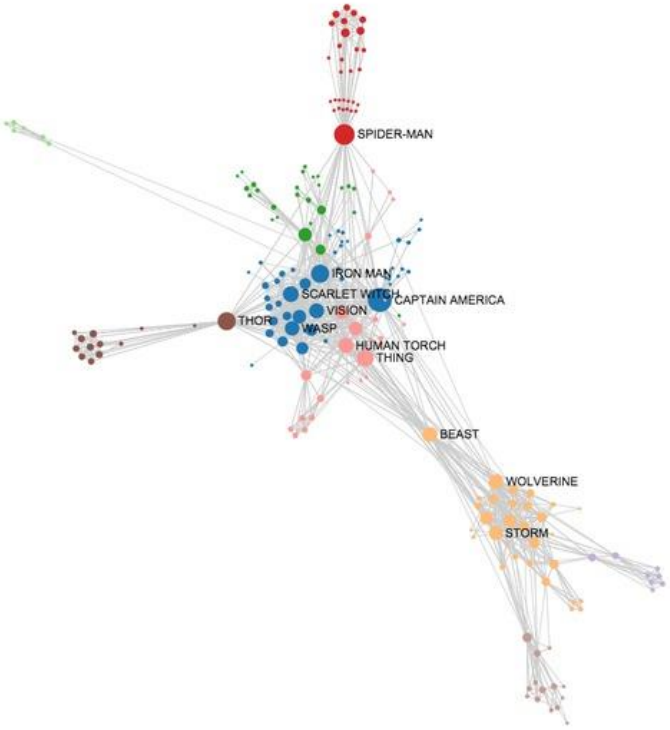


Figure 23 - Clustering Iteration 2

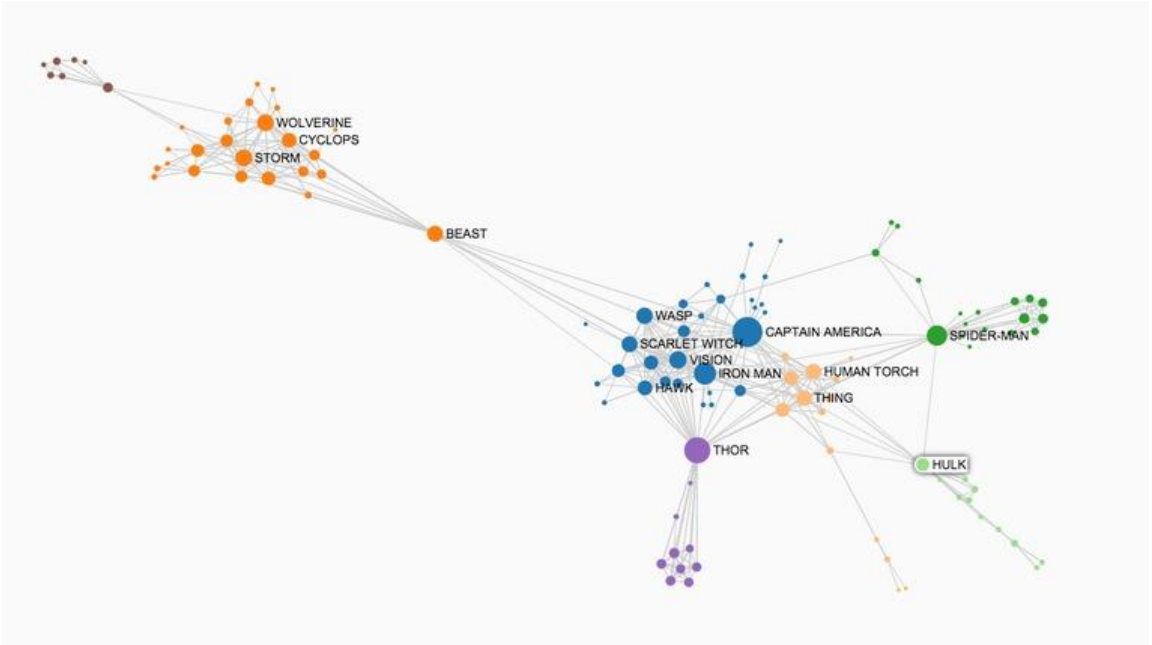


Figure 25 - Clustering Iteration 3

3.3 Collaborative Filtering

In this approach the traditional "Collaborative filtering" has been modified to generate user specific recommendations. The algorithm asks the user for a specific movie ratings on which he/she wants to get recommendations on, and then calculate the predictions for the movies according to their interests.

Here, we make predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption is that if a user A has the same opinion as a user B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a user chosen randomly.

The image below (from Wikipedia) shows an example of collaborative filtering. At first, people rate different items (like videos, images, games). Then, the system makes predictions about a user's rating for an item not rated yet. The new predictions are built upon the existing ratings of other users with similar ratings with the active user. In the image, the system predicts that the user will not like the video.

Getting and Processing Data

In order to build an on-line movie recommender we need to use Spark, and have our model data as pre-processed as possible. Parsing the dataset and building the model every time a new recommendation needs to be done is not the best of the strategies.
The list of task we can pre-compute includes:

- Loading and parsing the dataset. Persisting the resulting RDD for later use.
- Building the recommender model using the complete dataset. Persist the dataset for later use.

Loading and Parsing Datasets

No we are ready to read in each of the files and create an RDD consisting of parsed lines.

Each line in the ratings dataset (`ratings.csv`) is formatted as:

```
userId,movieId,rating,timestamp
```

Each line in the movies (`movies.csv`) dataset is formatted as:

```
movieId,title,genres
```

Were *genres* has the format:

```
Genre1|Genre2|Genre3...
```

The tags file (`tags.csv`) has the format:

```
userId,movieId,tag,timestamp
```

And finally, the `links.csv` file has the format:

```
movieId,imdbId,tmdbId
```

How to Make Recommendations?

When using collaborative filtering, getting recommendations is not as simple as predicting for the new entries using a previously generated model. Instead, we need to train again the model but including the new user preferences in order to compare them with other users in the dataset. That is, the recommender needs to be trained every time we have new user ratings (although a single model can be used by multiple users of course). This makes the process expensive, and it is one of the reasons why scalability is a problem (and Spark a solution). Once we have our model trained, we can reuse it to obtain top recommendations for a given user or an individual rating for a particular movie. These are less costly operations than training the model itself.

3.4 Content-Based Filtering

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content of the items and a user profile. The content of each item is represented as a set of descriptors or terms, typically the words that occur in a document. The user profile is represented with the same terms and built up by analyzing the content of items which have been seen by the user.

Several issues have to be considered when implementing a content-based filtering system. First, terms can either be assigned automatically or manually. When terms are assigned automatically a method has to be chosen that can extract these terms from items. Second, the terms have to be represented such that both the user profile and the items can be compared in a meaningful way. Third, a learning algorithm has to be chosen that is able to learn the user profile based on seen items and can make recommendations based on this user profile.

Exploration Strategies

The learning methods applied to content-based filtering try to find the most relevant documents based on the user's past behaviour. Such approach however restricts the user to documents similar to those already seen. This is known as the over-specialization problem. As stated before the interests of a user are rarely static but change over time. Instead of adapting to the user's interests after the system has received feedback, one could try to predict a user's interests in the future & recommend documents that contain information that is entirely new to the user.

A recommender system has to decide between two types of information delivery when providing the user with recommendations:

- Exploitation. The system chooses documents similar to those for which the user has already expressed a preference.
- Exploration. The system chooses documents where the user profile does not provide evidence to predict the user's reaction.

3.5 Technology

i. Operating System - Windows 10, Ubuntu 14.04 LTS
ii. Languages - R, Java, Python
iii. Environment - RStudio, IntelliJ IDEA, Enthought Canopy
iv. Build Tools - Maven, Apache Spark
v. System Requirements - Python 2.7+, JDK 7+, Git

CHAPTER 4 – PERFORMANCE ANALYSIS

4.1 Apriori Algorithm Analysis

The execution time of the algorithm significantly depends on the support count and confidence parameter.

As the support count is decreased the computation time of the algorithm increases since with the decrease in the support count there is a large increase in the number of large sequences in the result. The generation of rules does not count any candidate sequence that contains any subsequence which is not large. There exist some memory constraints although. If memory gets filled up, it is forced to count the last set of candidates generated even if the heuristic suggests skipping some more candidate sets. This effect decreases the skipping distance between the two candidate sets that are indeed counted and the efficiency henceforth decreases.

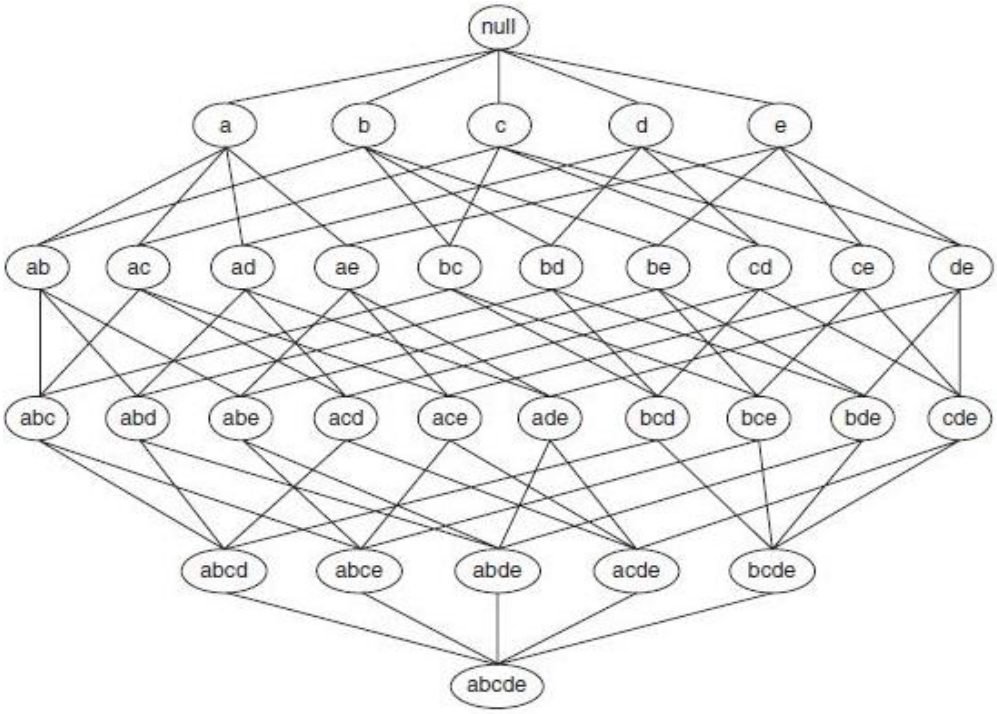


Figure 26 - Itemset Generation Lattice

I. Frequent Item-set Generation

A brute-force approach for finding frequent itemsets is to determine the support count for every candidate itemset in the lattice structure. To do this we need to compare each candidate against every transactions, as shown in the above figure. If the candidate is contained in the transaction, its support count is incremented. Such an approach can be very expensive.
Complexity: $O(NMw)$

N is the number of transactions,

$M = 2^k - 1$ is the number of candidate itemsets.

w is the maximum transaction width.

Ways to reduce the computational complexity of frequent itemset generation.

1. Reduce the number of candidate itemsets (M). The *Apriori* principle, described in the next section, is an effective way to eliminate some of the candidate itemsets without counting their support values.
2. Reduce the number of comparisons. Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set.

4.2 K-nearest Neighbor Analysis

I. k = 1 or Nearest Neighbor Rule

This is the simplest scenario. Let x be the point to be labeled. Find the point closest to x Let it be y. Now nearest neighbor rule asks to assign the label of y to x. This seems too simplistic and sometimes even counter intuitive. If it feels that this procedure will result a huge error, we are right – but there is a catch. This reasoning holds only when the number of data points is not very large.

If the number of data points is very large, then there is a very high chance that label of x and y are same. An example might help – Let's say we have a (potentially) biased coin. We toss it for 1 million time and you have got head 900,000 times. Then most likely your next call will be head. We can use a similar argument here.

II. k = K or k-Nearest Neighbor Rule

This is a straightforward extension of 1NN. Basically what we do is that we try to find the k nearest neighbor and do a majority voting. Typically k is odd when the number of classes is 2. Lets say k = 5 and there are 3 instances of C1 and 2 instances of C2. In this case , KNN says that new point has to labeled as C1 as it forms the majority. We follow a similar argument when there are multiple classes.

One of the straight forward extension is not to give 1 vote to all the neighbors. A very common thing to do is *weighted kNN* where each point has a weight which is typically calculated using its distance. For eg under inverse distance weighting, each point has a weight equal to the inverse of its distance to the point to be classified. This means that

neighboring points have a higher vote than the farther points.

It is quite obvious that the accuracy *might* increase when you increase k but the computation cost also increases.

III. Some Observations:

1. If we assume that the points are d-dimensional, then the straight forward implementation of finding k Nearest Neighbor takes O(dn) time.
2. We can think of KNN in two ways – One way is that KNN tries to estimate the posterior probability of the point to be labeled (and apply bayesian decision theory). An alternate way is that KNN calculates the decision surface (either implicitly or explicitly) and then uses it to decide on the class of the new points.
3. There are many possible ways to apply weights for KNN – One popular example is the Shephard's method.
4. Even though the naive method takes O(dn) time, it is very hard to do better unless we make some other assumptions. There are some efficient data structures like **KD-Tree** which can reduce the time complexity but they do it at the cost of increased training time and complexity.
5. In KNN, k is usually chosen as an odd number if the number of classes is 2.
6. Choice of k is very critical – A small value of k means that noise will have a higher influence on the result. A large value make it computationally expensive and defeats the basic philosophy behind KNN (that points that are near might have similar densities or classes) .A simple approach to select k is set $k = \sqrt{n}$

4.3 Evaluation Metrics

In recommender systems, most important is the final result obtained from the users. In fact, in some cases, users don't care much about the exact ordering of the list; a set of few good recommendations is fine. Taking this fact into evaluation, we could apply classic information retrieval metrics to evaluate those engines: 1. Precision 2. Recall and 3. F1-Score. These metrics are widely used on information retrieving scenario and applied to domains such as search engines, which return some set of best results for a query out of many possible results. For a search engine for example, it should not return irrelevant results in the top results, although it should be able to return as many relevant results as possible. Precision is the proportion of top results that are relevant, considering some definition of relevant for your problem domain. The Precision at 10 would be this proportion judged from the top 10 results. The Recall would measure the proportion of all relevant results included in the top results.

In a formal way, we could consider documents as instances and the task it to return a set of relevant items given a search term. So the task would be assigning each item to one of two categories: relevant and not relevant. Recall is defined as the number of relevant items retrieved by a search divided by the total number of existing relevant items, while precision is defined as the number of relevant items retrieved by a search divided by the total number of items retrieved by the search.

In recommender systems those metrics could be adapted hence; the precision is the proportion of recommendations that are good recommendations. And recall is the proportion of good recommendations that appear in top recommendations.

$$\text{Precision} = \frac{t_p}{t_p + f_p} \qquad \text{Recall} = \frac{t_p}{t_p + f_n}$$

Where t_p is the interesting item recommended to the user, f_p is the uninteresting item recommended to the user, and the f_n is the interesting item not recommended to the user.

In recommendations domain, a perfect precision score of 1.0 means that every item recommended in the list was good (although says nothing about if all good recommendations were suggested) whereas a perfect recall score of 1.0 means that all good recommended items were suggested in the list. Typically when a recommender system is tuned to increase precision, recall decreases as a result (or vice versa).

The F-Score or F-measure is a measure of a statistic test's accuracy. It considers both precision p and recall r of the test to compute the score: p is the number of correct results divided by the number of all returned results and r is the number of correct results divided by the number of results that should have been returned. It should interpret it as a weighted average of the precision and recall, where the best F1 score has its value at 1 and worst score at the value 0.

F- Score calculation using precision and recall:

$$\text{F-Score} = 2.\frac{\text{precision.recall}}{\text{precision} + \text{recall}}$$

In recommendations domain, it is considered a single value obtained combining both the precision and recall measures and indicates an overall utility of the recommendation list.

Evaluations are really important in the recommendation engine building process, which can be used to empirically discover improvements to a recommendation algorithm.

CHAPTER 5 – CONCLUSION

In the context of ever-increasing amounts of available information and data, it is difficult to know what information to look for and where to look for it. Computer-based techniques have been developed to facilitate the search and retrieval process; one of these techniques is recommendation, which guides users in their exploration of available information by seeking and highlighting the most relevant information. Recommender systems have their origins in a variety of areas of research, including information retrieval, information filtering, text classification, etc. They use techniques such as machine learning and data mining, alongside a range of concepts including algorithms, collaborative and hybrid approaches, and evaluation methods.

Having first presented the notions inherent in data- and information-handling systems (information systems, decision support systems and recommender systems) and established a clear distinction between recommendation and personalization, we then presented the most widespread approaches used in producing recommendations for users (content-based approaches, collaborative filtering approaches, knowledge-based approaches and hybrid approaches), alongside different techniques used in the context of recommender systems (user/item similarity, user/item relationship analysis and user/item classification). These concepts were then illustrated by a discussion of their practical applications in a variety of domains. Finally, we considered a number of different techniques used in evaluating the quality of recommender systems.

However, systems and techniques need to evolve over time, with the aim of improving performance, speed and proximity to the expectations or requirements of users. Several challenges remain to be met, for example:

- The improvement of collaborative filtering techniques, using more data sources (metadata or tagging data, demographic information, temporal data, etc.) or combining techniques that have yet to be used together.
- The volume of available data is constantly increasing and recommender systems encounter performance issues. They need to provide high-quality recommendations in record time in spite of this increase in data volume.

- Multi-criteria recommendation approaches are undergoing significant developments. The exploitation of multi-criteria scores, which contain contextual information, would be useful in improving recommendation quality.
- Contextual approaches (also mentioned briefly in this book) aim to take account of an individual's emotional context: for example, a person in love will find a romantic film more relevant than someone in a different emotional situation.
- Recommender systems use user data (profiles, etc.) to generate personalized recommendations. These systems attempt to collect as much data as possible. This may have a negative effect on user privacy (the system knows too much). Systems, therefore, need to make selective and reasonable use of user data and to guarantee a certain level of data security (non-disclosure, etc.).

In conclusion, recommender systems still need to respond to a number of different challenges. Developed in the context of various research areas, they take different forms and transcend multiple disciplines. This field of research needs to remain as wide as possible in order to identify the most appropriate techniques and approaches for each specific application.

Future Scope & Applications

(i) Location Variable

With increasing smartphone penetration and the advent of "Internet of Things" devices, the use of location in recommender systems and machine learning techniques is only bound to increase. Location based services exist today, but in isolation. Services need to become more responsive according to the context of the user. Ex: Google Maps, Yelp, Foursquare, etc.

A Location Aware Recommender System tackles a problem untouched by traditional recommender systems by dealing with 3 types of location-based ratings, namely:

- Spatial ratings for non-spatial items,
- Non-spatial ratings for spatial items, and
- Spatial ratings for spatial items

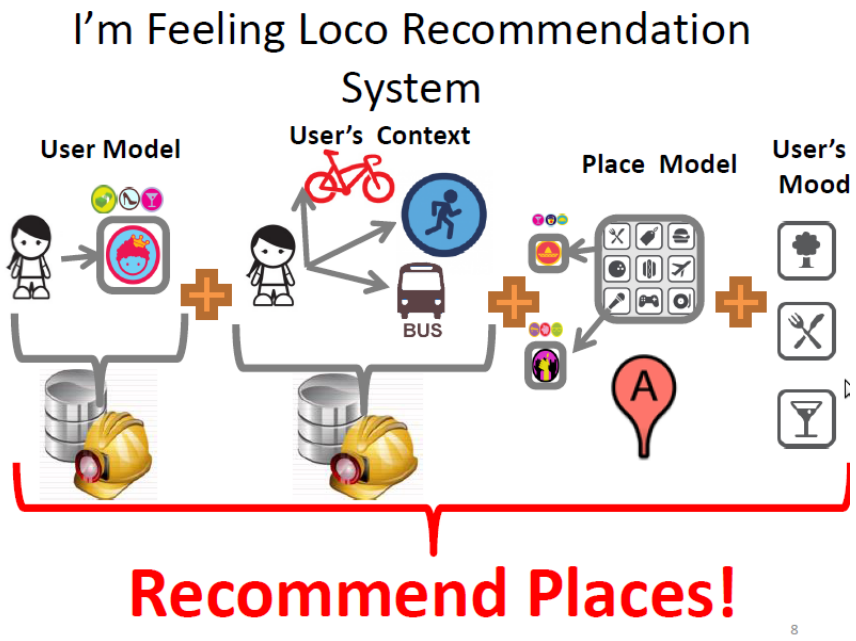


Figure 27 - Location Based Recommendations

(ii) Personal Assistants and Artificial Intelligence

As our devices start collecting more and more personal data of users and big data analytics becomes more advanced, the devices will be able to *decipher* our *'daily living patterns'* in real time. This can only happen via IoT devices and a central *inflexion* point. This inflexion point would be the voice assistant or personal assistant, becoming ever ubiquitous in our life and offering suggestions based on user habits and behaviour.

Intelligent assistants will be able to gather data via a knowledge graph, i.e., via the semantic web where connections to different data entities will be already available. These assistants need to be cross-platform and open to all kinds of third party apps in order to become truly ubiquitous. Also, they would need to be a *holding platform*, i.e.,

→ They need to be baked into the OS, and

→ Be the social networking hub of the user – a digital living room

This would eventually lead to the creation of Artificial Intelligence through deep neural networks where the assistants will be able to tell our needs through our daily habits and sense our mood via real time sentiment analysis of our social networks and communication behaviour. Ex: Siri, Google Now, Cortana, Facebook M.

References

[1] Yueping Wu and Jianguo Zheng, "A Collaborative Filtering Recommendation Algorithm Based on Improved Similarity Measure Method", *Proc. of Progress in informatics and computing*, 2010, pp.246-249

[2] Thangavel SK, Thampi NS, Johnpaul C, 'Performance Analysis of Various Recommendation Algorithms Using Apache Hadoop and Mahout', International Journal of Scientific & Engineering Research, Volume 4, Issue 12, December 2013

[3] Dhoha Almazro, Ghadeer Shahatah, Lamia Albdulkarim, Mona Kherees, Romy Martinez and William Nzoukou, "A Survey Paper on Recommender Systems", *Proc.of ACM SAC*, 2010, pp.1-11

[4] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl, "Evaluating collaborative filtering recommender systems", *Proc ACM Transactions on Information Systems*, 2004, pp.5-53

[5] J Liu, P Dolan, ER Pedersen, 'Personalized News Recommendation Based on Click Behavior', Google Inc.

[6] Das AS, Garg A, Datar M, 'Collaborative Filtering', Google Inc.

[7] Greg Linden, Brent Smith, and Jeremy York, 'Amazon.com Recommendations – Item-to-Item Collaborative Filtering', Amazon.com

[8] Sarwar, B. and Karypis, "Item-based collaborative filtering recommendation algorithms*", Proc 10th International World Wide Web Conference*, 2001, pp.285295.

[9] Saretto CJ, 'Contextual based information aggregation system', Microsoft Technology Licensing, Llc

[10] Levandoski, JJ,Sarwat M, Eldawy A, Mokbel MF, 'LARS: A Location-Aware Recommender System', Microsoft Research WA, USA, University of Minnesota, MN, USA

[11] Gross JN, 'Social Network Site Recommender System & Method', Facebook Inc

[12] Torrens M, Ferrera P, 'User to User Recommender', Apple Inc.

[13] Schafer,J.B., Konstan,J.A., and Riedl,J. 'Recommender Systems in E–Commerce', Data Mining and Knowledge Discovery

[14] Jorge Aranda, Inmar Givoni, Jeremy Handcock, Danny Tarlow, 'An Online Social Network-based Recommendation System', Department of Computer Science, University of Toronto, Toronto

[15] Jianming He and Wesley W. Chu, 'A Social Network-Based Recommender System (SNRS)', Computer Science Department, University of California, Los Angeles, CA