

Applied Recommender Systems — A Report

Lakshit Verma

March 30, 2024

Contents

1	Introduction	2
2	Introduction to Recommendation Systems	2
2.1	Types of recommendation engines	2
2.1.1	Market basket analysis	2
2.1.2	Content-Based Filtering	2
2.1.3	Collaborative-Based Filtering	3
2.1.4	Hybrid Systems	3
2.1.5	ML clustering	3
2.1.6	ML classification	4
2.1.7	Deep Learning	4
2.2	Applications	4
2.2.1	Popularity	4
2.2.2	Buy Again	5
3	Content-Based Recommender Systems	5
4	Collaborative Filtering	5
5	C Filtering using Matrix Factorizing, Singular Value Decomposition, and Co-Clustering	5

1 Introduction

This report is a short summary of the book **Applied Recommender Systems with Python**. It covers chapters 1, 3, 4 and 5 of the text respectively. In the initial chapter, we get a brief introduction to the world of recommendation systems and their types. In the subsequent chapters, we get a deeper dive into three methods of recommendation—Content-Based, Collaborative Filtering, and the methods used to achieve and improve on Collaborative Filtering.

2 Introduction to Recommendation Systems

recommendation systems are typically built for one purpose—to maximize revenue by enhancing the user’s experience and maximize their time spent on the site. To build a recommendation system, the most crucial element is user feedback. This comes in two forms, **explicit** and **implicit**.

- **Explicit Feedback:** It is feedback the user knowingly and explicitly puts on a product. Examples of this include likes, dislikes, star ratings, and reviews.
- **Implicit Feedback:** This is the feedback that is generated unconsciously, through the revealed preferences of the user. These can include links clicked on, pages visited, video watch time, etc.

2.1 Types of recommendation engines

There are several methods of creating a recommendation engine. The ones given in the text are as follows:

1. **Market basket analysis (association rule mining)**
2. **Content-based filtering**
3. **Collaborative-based filtering**
4. **Hybrid systems**
5. **ML clustering**
6. **ML classification**
7. **Deep learning and NLP**

2.1.1 Market basket analysis

This is method most often used by retailers to predict the popularity of a given item. It works through identifying pairs of items that are often put together.

A few important terms are used in context of this system, and one of them is **Association rule**. Their purpose is to identify and symbolize strong relationships between items. They are written in the form **{antecedent -> consequent}**. For example, take **{bread -> jam}**, which means in plain English “**There is a strong relationship between customers who bought bread and jam in the same purchase**”.

Support is the relative frequency of an association rule displaying. **Confidence** measures the reliability of the rule, where a confidence of 0.5 indicates the items in question were purchased together 50% of the time. Finally, **Lift** is a measure of the ratio of the expected support vs. the support if two rules are independent. A lift value close to one means the rules are independent, and lift values over 1 indicate more and more correlation between the two rules.

2.1.2 Content-Based Filtering

Content-based filtering method is a recommendation algorithm that suggests items similar to the ones other users have previously selected or shown interest in.

Take the example of Netflix. The popular streaming site saves all user viewing information in a vector-based format, known as the **profile vector**, which contains information on past viewings, liked and disliked shows, most frequently watched genres, and so on. Then there is another vector that stores all the information regarding the titles (movies and shows) available on the platform, known as the **item vector**. It stores information like the title, actors, genre, language, length, crew info, synopsis, etc.

The content-based filtering algorithm uses the concept of cosine similarity. In it, you find the cosine of the angle between two vectors — the profile and item vectors in this case. Suppose A is the profile vector and B is the item vector, then the (cosine) similarity between them is calculated as follows:

$$\text{sim}(A, B) = \cos \theta = \frac{A \cdot B}{\|A\| \|B\|}$$

This outcome always ranges between -1 and 1, and is calculated for multiple item vectors, keeping the profile vector constant. They are then ranked in descending order of similarity, and have one of two following approaches applied for recommendations:

- **Top-N approach:** The top N movies are recommended, where N limits the number of titles recommended.
- **Rating scale approach** A limit on the similarity value is set, and all the titles satisfying that threshold are recommended.

Other methods such as **Euclidean Distance** ($\sqrt{(x_1 - y_1)^2 + \dots + (x_N - y_N)^2}$) and **Pearson's correlation** are also utilized in select cases.

The critical flaw of such a system is the fact that all suggestions emerging from this end up falling into the same sort of product “category”, making it feel formless and repetitive.

2.1.3 Collaborative-Based Filtering

In this, a user-user similarity is considered along with item similarities, to address the issues with simple Content-Based filtering.

The process of finding similarities is much the same as that of Content-Based filtering, but the engine then recommends titles that the user has not watched but other users with the same interests have. There are two kinds of Collaborative-Based filtering algorithms.

- **User-user collaborative filtering:** Here, you find user-user correlations and offer recommendations based on what similar users chose before. Despite its effectiveness, it is highly compute-intensive and its use in large-scale databases is therefore discouraged.
- **Item-item collaborative filtering:** This involves finding similarities in item-item pairs instead of user-user pairs. This is far less computationally demanding, at the cost of less accurate results.

2.1.4 Hybrid Systems

A hybrid system offers the best of both worlds— combining both content-based and collaborative-based systems, drawing power from another when one fails to produce desirable results. They can be implemented in the following ways:

- Generating recommendations separately by using content- and collaborative-based systems and merging them subsequently.
- Adding features of the collaborative method to a content-based recommender engine.
- Adding features of the content method to a collaborative-based recommender engine.

Studies consistently show that hybrid recommender engines generally perform better, faster and provide more reliable recommendations.

2.1.5 ML clustering

Applying the novel field of Machine Learning to the recommendation systems seems like the logical next step. ML methods are of two types: supervised and unsupervised. Clustering is the unsupervised method, meaning it finds patterns in data sans any human intervention in labelling the data. It is the process of grouping objects into clusters, and generally an object belonging to a cluster is more similar to the objects inside the cluster than the objects outside of it.

Clustering based methods are usually implemented when there is little user data to go by. If a user is found to be similar to a cluster of users, the user is added to that cluster. Users inside the cluster all share specific tastes, and recommendations are provided according to that.

Some popularly used clustering algorithms are:

- **K-means clustering**
- **Fuzzy mapping**
- **Self-organizing maps (SOM)**
- **Hybrids of two or more techniques**

2.1.6 ML classification

In a classification based system, the algorithm uses features of both the items and users to predict a user's affinity towards a given product. One application of this is the buyer propensity model.

Some flaws of classification-based systems are:

- Collection of data is tedious.
- Its classification is challenging, and again, time-consuming.
- Training the models to function in real time is difficult.

2.1.7 Deep Learning

Deep Learning and Deep Neural Networks (DNNs) are a more powerful form of Machine Learning. They work especially well on unstructured data such as text, images, and video.

A few DL-based systems are:

- **Restricted Boltzmann**
- **Autoencoder based**
- **Neural Attention based**

2.2 Applications

Here, we look at real examples of recommendation systems being constructed using Python and sci-py.

2.2.1 Popularity

This is the simplest form of recommendation— to sort products based on a measure of popularity (views, downloads, likes, etc.)

We import the required libraries and data first.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('data.csv',encoding= 'unicode_escape')
```

Cleaning up the data by dropping NULLs and values with no description, we finally sort the most popular items.

```
global_popularity=df.pivot_table(index=["StockCode","Description"],
values="Quantity", aggfunc="sum").sort_values(by="Quantity", ascending=False)
print("10 most popular items globally....")
global_popularity.head(10)
```

To calculate the most popular items by country, we use the following code:

```
# Popular items by country
countrywise=df_new.pivot_table(index=["Country","StockCode","Description"],
values="Quantity", aggfunc="sum").reset_index()

# Vizualize top 10 most popular items in UK
sns.barplot(y="Description", x="Quantity",
            data=countrywise[countrywise["Country"] == "United Kingdom"]
            .sort_values(by="Quantity", ascending=False).head(10))
plt.title("Top 10 Most Popular Items in UK", fontsize=14)
plt.ylabel("Item")
```

2.2.2 Buy Again

This is another simple system that sorts items by number of repeated instances, and recommends the ones at the top.

```
from collections import Counter

def buy_again(customerid):
# Fetching the items bought by the customer for provided customer id
items_bought = df_new[df_new["CustomerID"] == customerid].Description

# Count and sort the repeated purchases
bought_again = Counter(items_bought)

# Convert counter to list for printing recommendations
buy_again_list = list(bought_again)

# Printing the recommendations
print("Items you would like to buy again :")

return(buy_again_list)
```

Using the function on a specific user, say 1252 (function call `buy_again(1252)`), we will recieve a list of the most likely items the user is to buy, based on their previous purchase history.

3 Content-Based Recommender Systems

4 Collaborative Filtering

5 C Filtering using Matrix Factorizing, Singular Value Decomposition, and Co-Clustering