Author’s Name: Sajal Sinha

Course Title: Product Based Recommendation engine

Keywords: Data Science, Unsupervised Machine Learning, Recommender Model

21 August 2021

Product Recommendation Engine

Abstract:

We present a product recommender model that is based on collaborative filtering. In this model, we have a 2 million customer review and ratings of beauty related products sold on amazon. The results showed that our model can provide appropriate recommendations to users.

Introduction:

Recommender model as one of the MOST demanding algorithms in recent times mainly in product based companies. With growing customers and use of digital media, recommender system helps to boost an app’s/websites engagement with its users. In this project, we developed a recommender model by using dataset related to over 2 Million customer reviews and ratings of Beauty related products sold on Amazon’s website.

Problem Statement :

We have dataset related to over 2 Million customer reviews and ratings of Beauty related products sold on Amazon’s website, and we are to develop a product recommender model using different approaches.

The Dataset has 4 columns:

- UserId: Unique code given to each users.
- ProductId: Unique code given to each product.

- Rating: On basis of experience, users rate product from 1 to 5.
- Timestamp: digital record of the time of occurrence of a particular event.

Methodology :

1. Installing libraries and getting Data
2. Check for null values and outliers
3. Exploratory Data Analysis
4. Model Preparation
5. Different Model approach and selection of best Model
6. Hyper parameter tuning of model
7. Final Evaluation.
8. Working

Approaches:

1. **Popularity based recommendation Model:** Popularity based recommendation Model can be said as most basic type of recommendation model. To put in simple words, popularity based model works on concept of ‘*Popularity’ .* Users are recommended products which are popular and have high chances of likability. Products can be classified as popular can be in terms of ratings, views, likes, purchase count, shares etc. A product loved by 100 people is most likely to be loved by other 50 as well, this is what popularity model is about. It has lot of drawbacks, but can be useful in some cases, especially when we have less variety of products. One of its major drawback is that if a good product isn’t rated or a bad produced is rated by a lot if people it will suggested to most users which can have negative affect on company.
2. **Collaborative Filtering Recommendation Model:** Collaborative Filtering Recommendation Model is a type of which is based on Users Historical Preference.

Popularity based model uses products historical data or say app/websites data to suggest products, on the other hand CF based model use each users historical data to suggest products. We can say that there is a chance that in popularity based model it may suggest same products to many of its users, but in CF it will only happen if two users are similar. CF works on principle of Nearest Neighbours algorithm. Even in Nearest Neighbours we have two approaches, User Based and Item Based. Let's discuss about User Based. Suppose we have A and B as two users, and if A and B are similar users then if A watched m1, m2, m3 so B will most watch m3 after m1 and m2. Considering ratings, while different people may have different baselines when giving ratings, some people tend to give high scores generally, some are pretty strict even though they are satisfied with items. To avoid this bias, we can subtract each user's average rating of all items when computing weighted average, and add it back for target user, shown as below.

$$r_{ij} = \bar{r}_i + \frac{\sum_k Similaries(u_i, u_k)(r_{kj} - \bar{r}_k)}{number\ of\ ratings}$$

Two ways to calculate similarity are **Pearson Correlation** and **Cosine Similarity**.

$$Pearson\ Correlation : Sim(u_i, u_k) = \frac{\sum_j (r_{ij} - r_i)(r_{kj} - r_k)}{\sqrt{\sum_j (r_{ij} - r_i)^2 \sum_j (r_{kj} - r_k)^2}}$$

$$\mathit{Cosine\ Similarity} : \mathit{Sim}(u_i, u_k) = \frac{r_i \cdot r_k}{|r_i||r_k|} = \frac{\sum_{j=1}^{m} r_{ij} r_{kj}}{\sqrt{\sum_{j=1}^{m} r_{ij}^2 \sum_{j=1}^{m} r_{kj}^2}}$$

Basically, the idea is to find the most similar users to your target user (nearest neighbours) and weight their ratings of an item as the prediction of the rating of this item for target user. For Item-based CF, we say two items are similar when they received similar ratings from a same user. Then, we will make prediction for a target user on an item by calculating weighted average of ratings on most X similar items from this user. One key advantage of Item-based CF is the stability which is that the ratings on a given item will not change significantly overtime, unlike the tastes of human beings.

To implement an **item based collaborative filtering,** KNN is a perfect go-to model and also a very good baseline for recommender system development. **KNN** is a **non-parametric, lazy** learning method. It uses a database in which the data points are separated into several clusters to make inference for new samples.

KNN does not make any assumptions on the underlying data distribution but it relies on **item feature similarity**. When KNN makes inference about a movie, KNN will calculate the "distance" between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest neighbor movies as the most similar movie recommendations.

There are quite a few limitations of this method. It doesn't handle sparsity well when no one in the neighbourhood rated an item that is what you are trying to predict for target user. Also, it's not computational efficient as the growth of the number of users and products.

3. **Matrix Factorisation:** Since sparsity and scalability are the 2 biggest challenges for normal CF method, it comes a more advanced method that decompose the first sparse matrix to low-dimensional matrices with latent factors/features and fewer sparsity. that's Matrix Factorization.

Beside solving the problems of sparsity and scalability, there's an intuitive explanation of why we'd like low-dimensional matrices to represent users' preference. A user gave good ratings to movie Avatar, Gravity, and Inception. they're not necessarily 3 separate opinions but showing that this users could be in favor of Sci-Fi movies and there could also be more Sci-Fi movies that this user would really like . Unlike specific movies, latent features is expressed by higher-level attributes, and Sci-Fi category is one among latent features during this case. What matrix factorization eventually gives us is what proportion a user is aligned with a group of latent features, and the way much a movie fits into this set of latent features. The advantage of it over standard nearest neighbourhood is that albeit two users haven't rated any same movies, it's still possible to seek out the similarity between them if they share the similar underlying tastes, again latent features.

To see how a matrix being factorized, very first thing to know is Singular Value Decomposition(SVD). Supported by Linear algebra , any matrix R are often decomposed into 3 matrices U, Σ, and V. Continuing using movie example, U is an n × r user-latent feature matrix, V is an m × r movie-latent feature matrix. Σ is an r × r square matrix containing the singular values of original matrix, simply representing how important a selected feature is to predict user preference.

$$R = U\Sigma V^T$$

$$U \in IR^{n \times r}, \quad \Sigma \in IR^{r \times r}, \quad V \in IR^{r \times m}$$

To sort the values of Σ by decreasing definite quantity and truncate matrix Σ to first k dimensions(k singular values), we will reconstruct the matrix as matrix A. the choice of k should confirm that A is in a position to capture the foremost of variance within the first matrix R, in order that A is that the approximation of R, $A \approx R$. The difference between A and R is that the error that's expected to be minimized. this is often precisely the thought of Principle Component Analysis.When matrix R is dense, U and V might be easily factorised analytically. However, a matrix of movie ratings is super sparse. Although there are some imputation methods to fill in missing values , we'll address a programming approach to only accept those missing values and find factor matrices U and V. rather than factorizing R via SVD, we try find U and V directly with the goal that when U and V multiplied back together the output matrix R' is that the closest approximation of R and no more a sparse matrix. This numerical approximation is typically achieved with Non-Negative Matrix Factorisation for recommender systems since there's no negative values in ratings.

4. Surprise Library: Surprise is a Python scikit for building and analyzing recommender systems that deal with explicit rating data.

Surprise was designed with the following purposes in mind:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on documentation, which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
- Alleviate the pain of Dataset handling. Users can use both *built-in* datasets (Movielens, Jester), and their own *custom* datasets.
- Provide various ready-to-use prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++,

NMF), and many others. Also, various similarity measures (cosine, MSD, pearson…) are built-in.

- Make it easy to implement new algorithm ideas.
- Provide tools to evaluate, analyse and compare the algorithms' performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by scikit-learn excellent tools), as well as exhaustive search over a set of parameters.

The name SurPRISE (roughly :)) stands for Simple Python RecommendatIon System Engine.

Drawbacks:

Collaborative Filtering is faced with cold start. When a replacement item coming in, until it's to be rated by substantial number of users, the model isn't ready to make any personalized recommendations . Similarly, for items from the tail that didn't get an excessive amount of data, the model tends to offer less weight on them and have popularity bias by recommending more popular items.

It's usually an honest idea to possess ensemble algorithms to create a more comprehensive machine learning model like combining content-based filtering by adding some dimensions of keywords that are explainable, but we should always always consider the tradeoff between model/computational complexity and therefore the effectiveness of performance improvement.

Result:

While selection of model Popularity Model gave an RMSE of 1.3 whereas KNNwithMeans model gave 1.05 which was better than popularity based. Further using SVD we got RMSE as 1.01 which was the lowest and on further hyper parameter tuning we got RMSE as 0.87.

References:

- https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea
- https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26
- https://en.wikipedia.org/wiki/Collaborative_filtering
- http://surpriselib.com