Data Scientist – Recommender Engine Use Case

Author: Nilam Tathawadekar

Date: 04.03.2024

Github Link: https://github.com/nilam-t/bundle_recommendation/tree/master

1. ***Implement a solution of your choice for recommending product bundles e.g. rule-based, statistics, or ML-based solution. Please describe any reasoning behind your solution.***

   In this project, I have implemented an Associate Rule Mining (ARM) – a rule based machine learning algorithm for finding patterns and enabling product bundle recommendations. ARM discovers strong rules in database using different measures of associations. It discovers rules that determine groups of items that are connected. Given a transactions' database consisting of N products, rule is defined as an implication of the form $X \rightarrow Y$, where X, Y are two different set of items, also known as itemsets (Agrawal, 1993). Rules are made by searching data for frequent patterns and by using criteria such as support, confidence, lift, etc.

   $support(X,Y) = Freq(X,Y)/N$ → calculates the probability of observing X and Y together.

   $Confidence(X,Y) = Freq(X,Y)/Freq(X)$ → calculates the probability of observing Y when X is sold.

   $Lift(X,Y) = support(X,Y)/\big(support(X) * support(Y)\big)$. → When X is bought, the probability of buying Y increases by Lift times.

   Thus, ARM analyses the relationship between particular items to one another within the set. The popular choices among ARM include Apriori, FP-Growth, Eclat, CARMA algorithm, etc.

   The Apriori algorithm which is a widely used algorithm, works on a principle that if a set of items is frequent, then its subsets are also frequent. It computes 'frequent itemsets' i.e., itemsets whose support is greater than or equal to minimum support threshold. Even though simple, it is a memory intensive algorithm due to multiple scans over itemsets to count support. The dataset under study contains the information of 25900 unique transactions consisting of 4070 unique products of an online retail store. For the given dataset, Apriori algorithm failed to provide bundle-recommendation due to high memory requirements. Therefore, I consider a more efficient FP-Growth algorithm. It uses a data-structure called a Frequency Pattern tree (FP tree) to store the frequency information of the itemsets. FP tree stores the data in a more compact way thereby reducing the database scans. Therefore, FP-Growth is faster and more scalable alternative to the Apriori algorithm.

   The Github repository (https://github.com/nilam-t/bundle_recommendation/tree/master/recommender_system) contains the code to generate the bundle recommendations for a given product ID.

2. ***Provide a splitting to train and test datasets. Discuss possible different splitting criteria. What other splitting criteria would you choose if you could gather more features/data?***

The dataset can be divided into training and test dataset using `InvoiceDate` feature. Provided data consists of transactions from (2010-12-01) to (2011-12-09) i.e., 12 months of transaction data. As transaction data is temporal in nature, algorithm can be trained using data of first 9 months (2010-12 – 2011-08) and tested using the data of last 3 months (2011-09 – 2011-12). Thus, the data is divided 75% and 25% for training and testing respectively. This train-test split offers an opportunity to continuously evaluate the impact of recommending associated products to customers. (A/B testing)

Another possible train-test split criteria is using `CustomerID` feature. Assuming similar purchase patterns, we can divide the transaction data using `CustomerID`.

3. ***Discuss the size of the output list and how it can be decided per product***

Deciding optimal bundle size (or size of the output list) is a pivotal question in recommendation system to improve and optimize the revenues. The implemented FP-Growth method computes association rules based on the frequent itemsets found. Each rule computed using this algorithm consists (among others) of the following information:

1. Antecedents: The first product that is assumed to be sold first.
2. Consequents: The next product(s) that is(are) assumed to be sold after the first product.
3. Support: Probability of observing the consequents and antecedents together.
4. Confidence: Probability of observing the consequents when antecedents are sold.
5. Lift: When antecedents are sold, the probability of selling the consequents increases by a factor of lift.

The rules are pruned using `support` metric. In order to decide the bundle size per product, these pruned rules are used. We filter the rules associated with the given `product_id` (`rules['antecedent']== product_id`). The resulting data consists of multiple rules where antecedent is the given `product_id`.

Density-based: One of the simplest method to obtain the optimal bundle size is by computing density of different bundle sizes for the given product. This can be implemented by counting the occurrences of bundle sizes in the filtered rules. However, we can improve this recommendation by taking into consideration the `Lift` metric.

Lift-based: `Lift` metric helps us to identify the bundle sizes with higher probabilities. Therefore, I propose that bundle size can be decided by aggregating the lift of the filtered rules for given product. Next, a few examples comparing these methods for different product IDs from the given dataset are shown. Following figures compare the density-based (left-hand side) and lift-based (right-hand side) statistics for different bundle sizes for a given product ID. The highlighted (using red colour) bundle-size is the recommended output list size for that particular product.

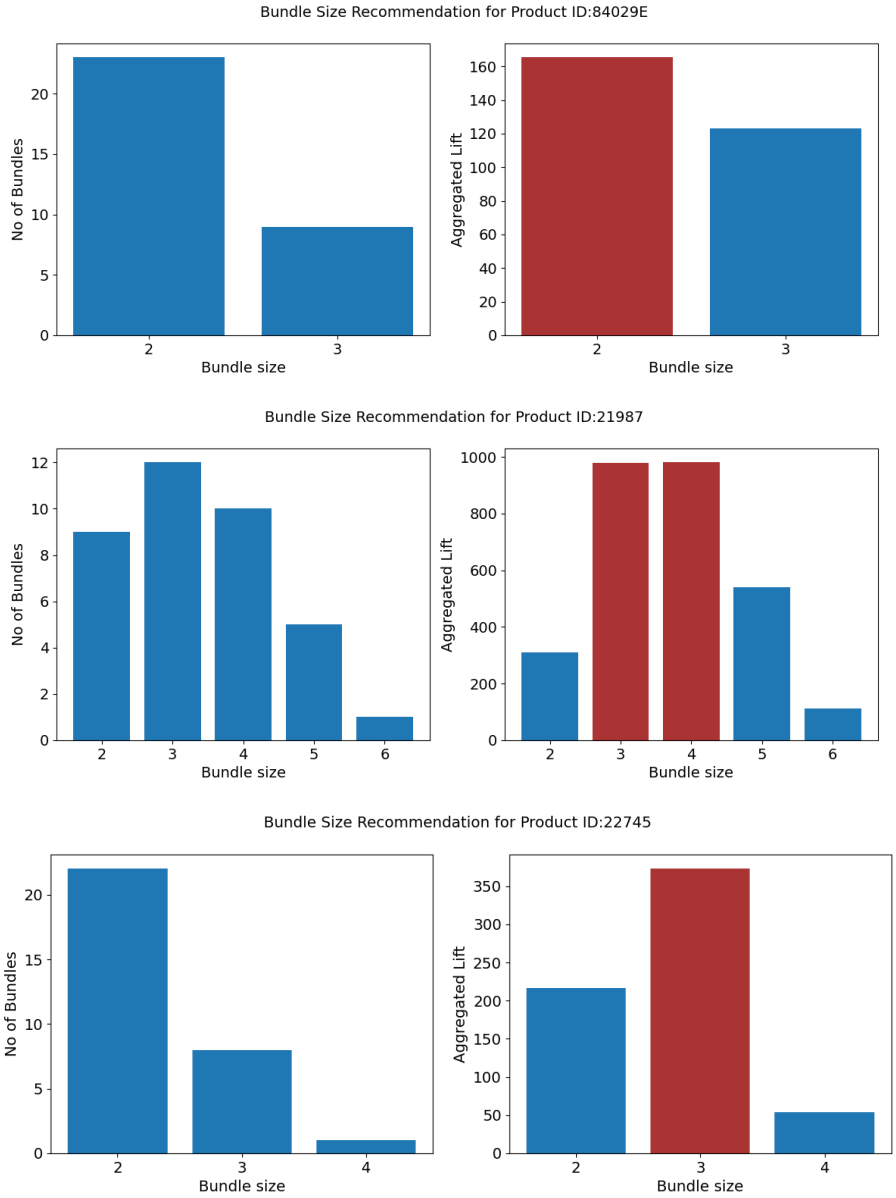

Figure 1: Comparison between density-based (on the left) and lift-based (on the right) statistics for different bundle sizes for a given product ID.

4. Discuss/implement any price computation per bundle e.g. the sum of products' prices

What makes bundle-recommender attractive to the user is 'ease of use' i.e., it enables users to quickly buy related items and accessories. It can be further enhanced by offering monetary benefits to the user. For example, waive the postage or shipping charges of the bundled items, offer discounts on the recommended products, promote the discounted/seasonal products in bundles. In addition to the sum of products' prices, highlighting the savings achieved if a customer buys the bundle might be of interest. E.g. 'You save *x* Euros', 'Get *x*% off' or 'No shipping charges' might be an effective way to promote the bundles. Therefore, in addition to the sum of products' prices, it would be beneficial to compute the discount or saving achieved per bundle.

5. ***How would you evaluate the business impact of the solution and share the outcome with the internal stakeholders?***

   In order to evaluate the impact of the solution, it would be beneficial to obtain the data with ground truth bundle recommendations. Then, we can split the data into train-test and evaluate the performance of the proposed methods using different metrics. One of the main metrics would be the `mean average precision at K` (MAP@K), which gives insights into how relevant the list of recommended bundles are. `Personalization` is another metric which can enable different recommendations across different users' i.e., each user getting more personalized recommendations based on the user history. `Diversity` across bundles is also an important metric from a business standpoint. Bundles with similar items can decrease the user's interest. Therefore, it is important to generate bundles with consistent yet diverse items.

   To evaluate how real customers react to the produced recommendations, A/B testing is important. It allows us to measure metrics such as Click-through rate (CTR), adoption and conversion measure, user engagement and behaviour, increase in sales and revenue, etc., that are important from business point-of-view.

6. ***Your bundle's code is a great success and the Frontend team wants to use it in production. Implement a simple Rest API to serve the bundles with an endpoint getting as a parameter a product ID and returning a list of products and the price for the whole bundle. Ideally, provide a Dockerized version of the implemented API.***

   The Github repository (https://github.com/nilam-t/bundle_recommendation/tree/master/rest_api_implementation) includes the Rest API implementation and its dockerized image.

References

Agrawal, R. a. (1993). Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, 207-216.