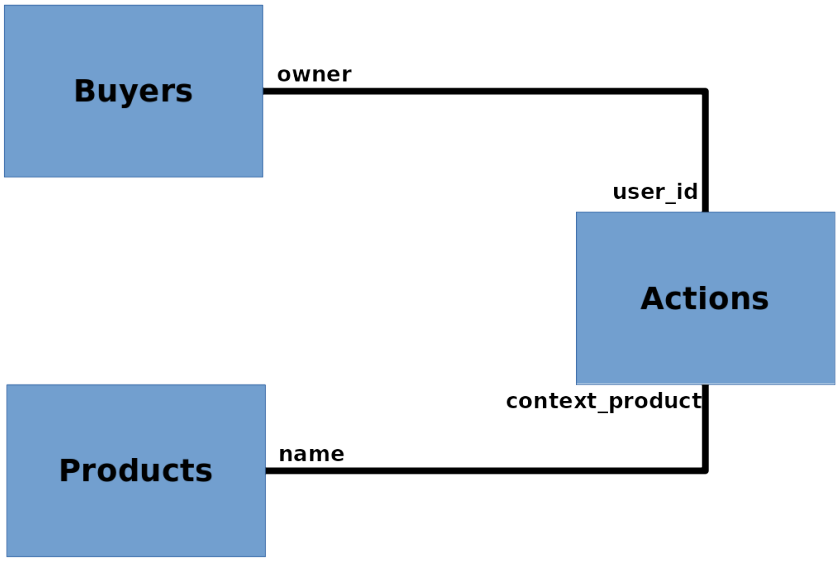
Hubba Product Recommendation

By: Joey Sham

How It Works

The 3 csv files (buyers.csv, actions.csv, products.csv) are loaded into pandas. Duplicates are dropped. After analyzing the data, I made an assumption that buyers' key is buyers["owner"], products' key is products["name"], and these can be joined to actions by actions["user_id"] and actions["context_product"] respectively. actions["context_product"] is the last value of actions["context_page_path"]. A diagram of that is shown below.



Before joining them, each dataframe has to be grouped by their specific key. There are product names that exists in "actions" but not in "products", and the details of those products are scraped from Hubba.

A numerous number of different actions could be performed by buyers on different product pages. If the user performed one of the following actions, it is deemed to be a positive action: "Liked Product", "Added Product To List", "Followed User", "Followed Brand", "Followed List", "saved product". A positive action can be seen as a score on how much the specific user likes each product. The dataframes are joined such that the final dataframe is of the form:

user_id	context_product	positive_action_count
53ff5739aebb450829000074	affect-health-drinking-chocolate	0
53ff5739aebb450829000074	whiskey-wicks-fresh-brewed-wood-wick-candle-vegan-soy-wax-coffee-scented-1	1
...	...	...

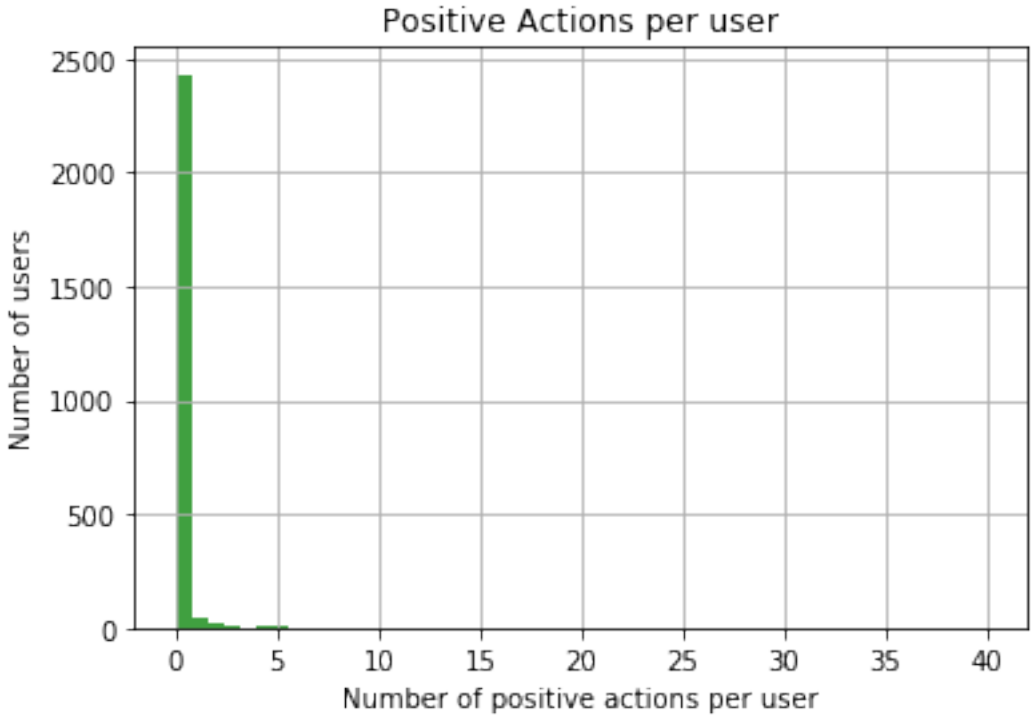
This effectively lists each user and their total positive interactions with each product. This dataframe is fed into SVD, which is the same recommendation method used by the winner of the Netflix contest. This is done by using scikit-surprise, which also allows prediction. The top 10 recommended items for each user is output and saved in a JSON file. The output results of this is displayed in a webapp written with Django.

Definition of Good

It is very difficult to define “Good” objectively. For the sake of this project, “Good” is anything that reacts positively to the what is displayed in front of it.

Good Buyer

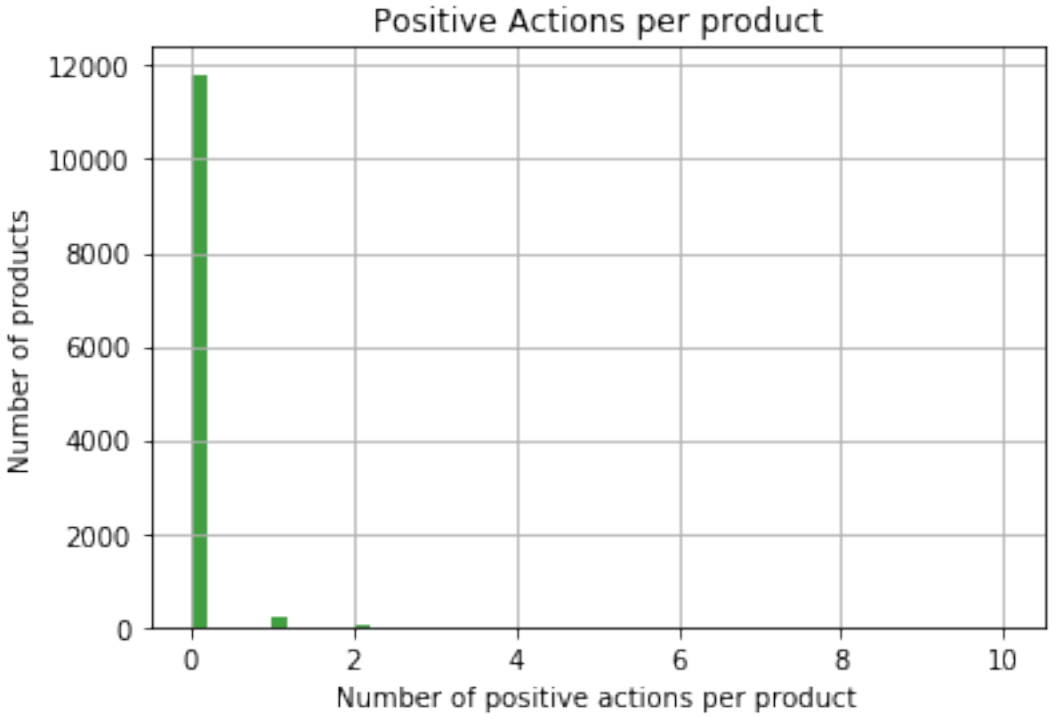
A good buyer is one who has a lot of interest in products. He/she may not necessarily be purchasing many products, but they are active in pursues of various products. This can be described numerically as the sum of all positive action counts across all products.



From the diagram above, it can be seen that most buyers do not have positive actions with products. These are users who mainly browse. The higher the positive actions count a user has, the better buyer they are considered. Any buyer with a positive actions count score 5 or greater can be considered a good buyer.

Good Product

A good product is one which has generated a lot of interests. Buyers may not necessarily be purchasing it, but they have interest in the product. This can be described numerically as the sum of all positive action counts across all users.



From the graph above, it can be seen that most user’s interaction with a product page aren’t positive actions. This is due to users casually browsing, rarely saving items or purchasing. Any product with positive actions count of 2 or greater is considered a good product.

Web Application

The result of the model is displayed in a web application, by wrapping the results in Django. The UI/UX is very bare, and is only enough to demonstrate the idea of recommendation for each user.

Short Comings

Most of the datapoints, whether it is a buyer or a product, does not have any positive actions associated with it. While diving into the dataset, it was discovered that many user actions in a page involve of searching, login, logout, and many generic actions that does not apply directly to the product. Loosening the definition of “positive actions” could result in a wider range of grades. For the buyers that has never performed positive actions (possibly new users), the current model returns a randomized set of products, when ideally it would return the most popular products.

Scraping Hubba products was performed to fill in missing data. However, the method of scraping was not foolproof, as there are still many items that were not scraped due to the fact that the scraper could not organically find the url of the product. Further investigations into this can fill in gaps of the data.

Future Improvements

The first most obvious drawback is the calculationn of positive actions acount. While this number is a good naive first step, it is far from an accurate representation of how the user feels about each product. Being able to accurately gage a user’s view on each product can allow more accurate predictions on what the user wants. One possible method is to give weights to various actions. For example, adding a product to a list can give a score of 1, while saving the product gives a score of 2. Removing it from a list reduces the score by -1.

Another improvement to the recommendation engine can be performed by tuning the SVD model. Due to time limitation, hyperparameters of SVD are not fully tuned. By improving this, users recommendations can be vastly improved.

The current recommendation utilizes a small portion of the data. Attempts were made to run it for the full dataset, but the limited resources caused the workstation to crash. Using the full dataset is necessary for the engine to improve its recommendations.

Moving to Production

The web application is currently hosted on the free tier of Heroku, supported by the free tier of Postgresql on Heroku. In order for the application to handle production load, it must be moved to a scalable cloud service. I would suggest moving both to AWS, with the server on an EC2 instance and the database on Redshift. ELB can be placed between each request and the EC2 instance, set such that when an EC2 instance has load exceeding 80%, another EC2 instance can be spun up and the load is shared. When load drops below 20%, an EC2 instance is shut down to save cost.

While this is hosted, the recommendation model can be rerun once a day, and update the database during off-peak hours and when it is not locked.