Grocery Store Application with Item Recommendation

## 

## COMP4601

## For

## Professor Anthony White

## 

## By

## Md. Tanvir Islam 100969657

**Abstract**

This project seeks to design a grocery shopping website’s catalogue such that customers receive recommendations based on the items in the basket, maximizing sales and customer satisfaction while playing into the psychology of impulse buying. Individual product data was extracted using a web crawler, which scraped all the products from a site called <https://fillmyfridge.ca/> and stored them MongoDB. This data was then piped into an android application depending on the search query made by the user using RESTful API calls. The grocery shopping app then used natural language processing to process this information and suggest recommendations to customers based off of the items already saved to their shopping basket, thereby maximizing sales.

**Introduction**

We are in the midst of a technological boom right now, catalysed by advancements in data collection; our exposure to incredible amounts of data increases every day, and with it so does our ability to measure, store, and predict the same-- giving us the capability to learn, experience and create without any limits. However, an exposure to such incredible amounts of data can be quite chaotic. With incredible amounts of data comes incredible amounts of choices and due to so many choices we can spend way too much time picking a certain data or worse pick something completely irrelevant to our needs. This is where recommender systems come in handy. They provide clarity in our decision to make meaningful connections out of huge amounts of information. Good recommender systems will know exactly what the user has in mind and from trillions of data points give the user exactly what they are seeking out. An excellent example of a cutting- edge recommendation system would be Amazon’s product recommendations. It inspired me to develop my own version of this technology, for use towards a smaller business ventures, which pose their own set of challenges.

Using my application, the user can search for a product in the store from all the existing product data extracted from fillmyfridge.ca. Upon receiving the products from the search result, the user can add the product to their cart or click on the cart for more details about the product. If the user decides to click on the card representing the product they will be shown further details about the product and other products recommended to them. These recommendations will be of two categories, substitutes and complements. Substitute goods are products that are similar to each other, thereby replacing one another on occasion, contingent on price and other factors. Meanwhile, complementary products are often bought in concert with one another. For instance, Coco Cola and Pepsi and substitutes, while chips and dip are complements.

**Background**

To build my system, I needed four main components, the client with which data is accessed, the server which serves the data, the data itself, and most importantly, the algorithm with which the recommendations could be made.

From thousands of products in the database the most relevant products need to be given back to the user according to their search; in order to do so I used a powerful indexing library called Lucene. Lucene is a full-text search library in java which makes it easy to add search functionality to an application or a website. It adds content to a full text index. It then allows you to perform queries on these indexes returning the most relevant documents depending on the query. Lucene is extremely fast because instead of searching texts directly it searches an index instead. It is similar to searching contents in a book through it’s indexes rather than going through every single page in a book looking for a word. This type of index is called an inverted index, because it inverts a page-centric data structure (page with words) to a keyword-centric data structure (words found in pages).

It is very important for a business to find the association rules between products that are bought together frequently. If the association rules can be found between products that are bought frequently together it can potentially increase the sales of a company, by better advertisements, discounts on one out of the two products or even create new products which combines the complementary products. The Apriori algorithm is a very good association rule analysis algorithm which I have used for my complementary product recommendations.

The apriori algorithm measures the association rules using support and confidence quantifiers. The support of a set of items determines how popular that set is compared to all the available sets. It is the ratio of the frequency of the item set to all other item sets. The confidence says how likely item-b is purchased when item-a is purchased. To clarify the statement let’s say we have the following transaction sets {milk, cereal, break}, {milk, cereal}, {milk, butter}, the confidence of purchase of milk when cereal is bought is expressed as {milk -> bread} and the confidence is 2/3.

It is also very important to recommend user products that are similar to each other, for example, if a user wants to buy chocolate flavoured ice cream they may also want to try out other flavours like strawberry. In order to make such recommendations of similar products I use natural language processing on Lucene search queries in order to find the most relevant products similar to the current item for recommendation.

**Related Work**

For recommendation of products, several other algorithms other than Apriori could be used: one example would be Item-to-Item collaborative filtering, an algorithm used by Amazon itself [1]. In Item-to-item collaborative filtering a table is created which consists of item pairings that customers tend to purchase together. This table is looked up when a client clicks on a new product they are purchasing to determine the other product that would go best with this product.

**Methodology**

A web crawler written in Java was used to scrape through all the product catalogues from fillmyfridge.ca and a total of 5521 products were extracted from the website. Information from a product catalogue page was carefully extracted to facilitate the indexing and searching of user’s product requests, each product was stored in MongoDB database with a unique id. The model that represented each product had the following attribute unique document identifier, URL, title, price, category of the product, product image URL, the date that the product was extracted, and all the metadata related to the web page that the product was extracted from. Each model and all it’s attributes were indexed using Lucene. Indexing every single document was necessary to allow fast retrieval of documents from the client side.

The client side for this project is an android application written in React Native. The app provides the following functionalities, searching products (Figure 1), viewing details about a product returned from a search result which further shows recommendations for similar products and complementary products (Figure 3), adding products to a shopping cart, and checking out and completing a transaction (Figure 2).

The data is piped to the client side using a server written using the Django Framework in Python. All the communication between the client and the server is using RESTful API calls. The client side is provided with a search bar where the user can type in their product query, which gets sent to the server using a GET HTTP request. The server then returns all relevant product details in a JSON object. The data inside the JSON object then gets used by the client side to display the results of the search. To display more details about the product another GET HTTP request is made to the server to display further information about the product. When the user completes a transaction a PUT HTTP request is made which sends a JSON object consists of all the product id in the transaction which gets saved inside MongoDB.

The product extracted from fillmyfridge.ca did not have enough information about the product other than the model attributes listed. This resulted in some queries providing with no results. For example, if a user searched for orange, no results would show up. To fix this problem every single query sent by the user was modified by including a plural version of each word in the query, so a query for ‘orange’ would become ‘orange oranges’, similarly queries with plural words also were attached with singular version of the words.

The highlight of this project is the recommendations given to user for a product which they are viewing. These recommendations are of two categories, substitutes and complements.

**Substitute Recommendation:**

The database contains several products which are similar or identical to each other. These products may be the same type but from different brands or may be in different flavours. I use two steps to get all the substitutes. First, to search the name of the product in Lucene. The results of the search will contain several other products which are completely unrelated to the product. Since all the products in the database are labelled with a category, I simply discard any product that does not belong in the same category as the product for which I am getting the substitutes. In the second step I extract all the noun words from the title of the product that I queried Lucene with. Separating the noun from the product name is crucial as products can be of the same category but can be completely different. For example, if you have a product called green peas which belongs to the vegetable category, peas are not the only thing that will get returned but everything that has the adjective green will also get returned. To fix this problem each remaining product’s title is checked with the list of noun words that the current product has. If the remaining product’s title has words that match 90% of the letters in the noun words, then the products are similar (all the noun words with 90% accuracy needs to be in the remaining product list). This final filter gives back the most relevant list of similar products: for instance, ‘wheat’ and ‘weat’, and ‘weat’ is slightly misspelled 90%, similarity score makes sure that it is still returned in the search results.

**Complement Recommendation:**

To generate complement recommendations for a product I used the Apriori algorithm. It is used to find interesting relationships between items in a large data sets. The hidden relationships are then expressed as a collection of association rules and collection of items that are frequently bought together. I use the support and confidence quantifiers generated from the Apriori algorithm to find relationships between the products. In order for this to work I needed transactions. Generating thousands of transactions would prove time inefficient if done by an individual.

To solve this particular problem I used a dataset of transactions [4] to generate transactions into the database on which apriori algorithm could be applied. This dataset contains 9835 transactions. The dataset is in a csv file format and each row represents a transaction and each column an item. The data in the csv file was parsed and a list was created which contained all unique items in the csv. For every item in the list of unique csv item a product from my database needed to represent that item. For every unique item a query needed to be sent to Lucene index to retrieve the products related to this item. The result of the products returned can include a lot of irrelevant products which has nothing to do with the search query. To remove irrelevant products I determine the category that the query item is most likely to belong to.

Every document returned by Lucene contains a score. I create a python dictionary where the key is a category and the value is the total of all the Lucene document score that each document belonging to that category scored. Using this score, I determine the category that the item in the transaction is most likely to belong to. From the all the documents returned by Lucene I discard all the documents that do not belong to the classified category and randomly pick a product from the classified category. This randomly picked product from the classified category will represent the unique item in the dictionary. Similarly, I find all my own product representations of the items in the database. If a there are absolutely no product representations found from the database that product is removed from each transaction in the csv file.

It is very important for each unique item in the csv to be represented by another unique item in our database. If the same product is not used to represent Apriori will output incorrect outputs as the frequency of the transactions will be low. With all the transactions now generated in the database I feed the data into an Apriori implementation and get the associations with a support value of 0.003 and a confidence value of 0.2. The support and confidence was determined through trial and error.

With the associations computed for a given product complementary products can be found. When a GET HTTP request is sent to the server for a list of complementary products the server computes does the Apriori on the transactions gives out all the results of associations with a minimum confidence value of 0.2 and support of 0.003. I then look up these values to check if the product I queried can be found in these associations. If it is found, then I get list of all the products that it was associated with and return those products. If a product does not exist in the apriori result I generate a list of similar products for which I am finding the complementary products for using my similar products algorithm. I then look up each product from the list of similar products in the results of the Apriori algorithm, if one of them matches then that products complementary products are returned.

**Discussion**

The goal of the project was to create an online grocery store application with a recommendation system-- which it does achieve, however there is a lot of room for improvement. The project’s biggest weakness is its hunger for data. The biggest challenges I faced was to create a recommendation shop for a store that has nothing to recommend as there are no real and meaningful transactions in the database. Simulation of real transaction is difficult without an external source of data. The product catalogue extracted from the crawler will not always contain all the products which exists in the external transaction dataset. This results in creating substitute products which does not really reflect the real users buying patterns, this may provide faulty recommendations for complementary products.

However, the project’s real strength lies in its search, interface and recommendation of similar items. The product catalog extracted from fillmyfridge.ca only had the title of the product, while the price and category it belonged to as texts that could be indexed using Lucene which made it difficult to retrieve relevant products due to the lack of enough text in the index itself. For example, a query with the text ‘green tea’ would return results for not only every document which included tea but also documents which had the word green in it, so finding all similar products related to tea would return irrelevant results. The strength of the project really lies in finding products that are alike by modifying search queries with natural language processing by identifying the nouns and adjectives and emphasizing the query with the noun part when looking up products in the Lucene index despite the lack of descriptive details about the product.

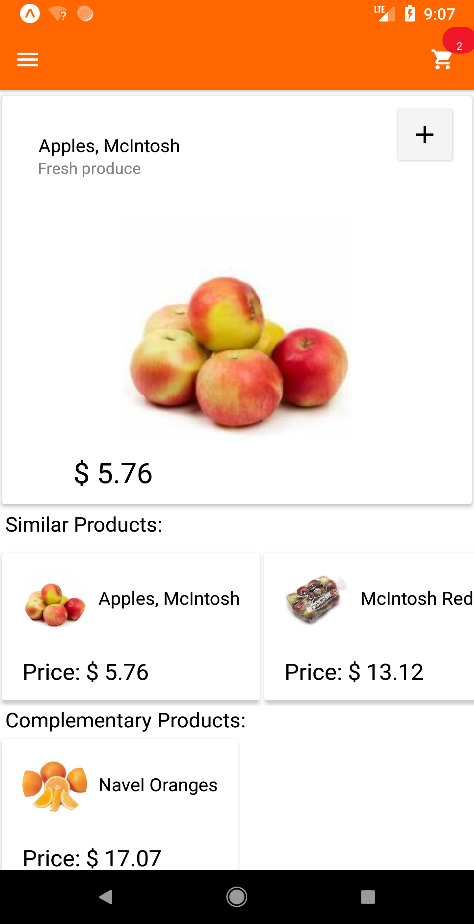
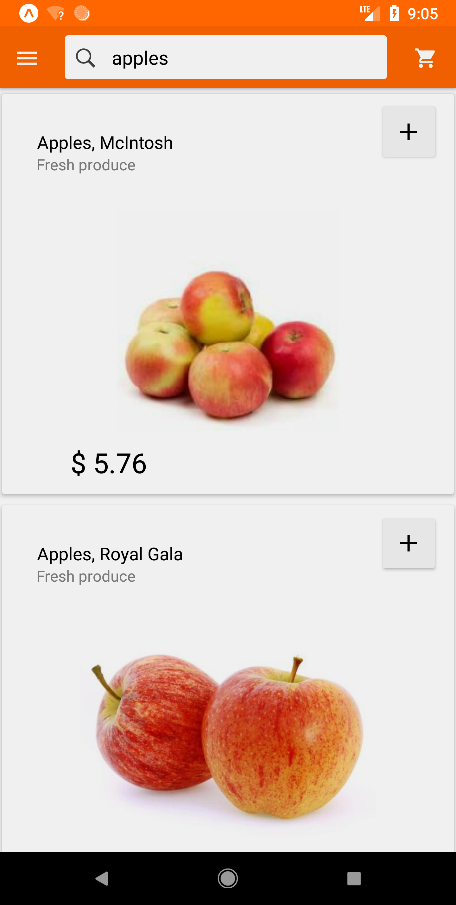
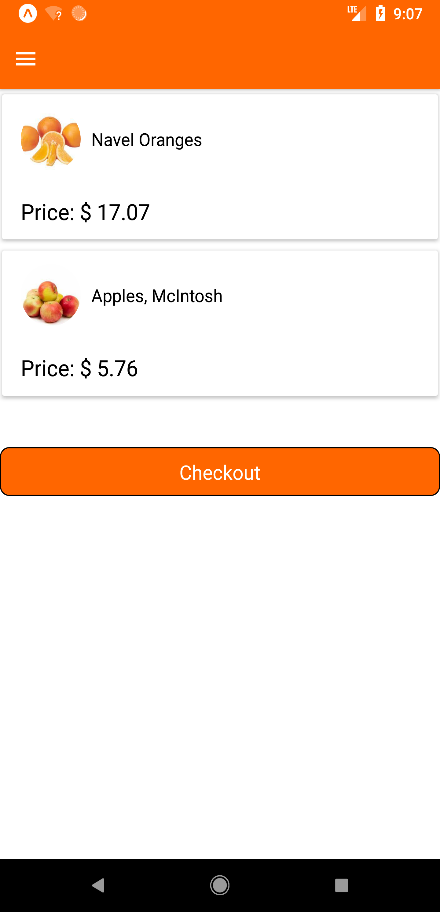
**Conclusion and Future Work**

In the future I would like to improve my recommendations by adding an extra recommendation for a product just as the user is about to checkout with their items in their cart. This recommendation is going to use a user based collaborative filtering algorithm. The current items in the cart will be converted into a 1 x n vector where n is the total number of products in the database. From n products, products that exists in the cart will be represented by 1 and 0 otherwise. Transactions that have already been completed by other users will also have this vector representation of the transaction. The cosine similarity between the checkout vector and every other transaction vector is calculated. A cosine similarity value greater than 0.5 will indicate that the transactions are very similar to each other. A matrix will be created using the checkout vector and all other similar vectors and user based collaborative filtering will be done on this matrix to find more items that the user can purchase with his current selection of items.

In the future I aim to introduce rating system to the items and create a user base where user’s can rate that they have purchased. Adding these two functional features to the application will provide a greater opportunity to improve product recommendations as algorithms such as item based collaborative filtering can then be implemented.

My ability to come up with these insights was directly impacted by the fact of putting all the work in towards the app as it stands right now; it has allowed me to understand that app development is bound to be an ongoing process, bounded only by what I know, and influenced by my own ability to learn from my mistakes, as well as from existing research in the field.

**Figures:**

****

**Figure 1. Home Page with Search Figure 2. Checkout Page Figure 3. Recommendation Page**

**References:**

1. <http://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf>
2. <http://franklinwillemen.com/machine-learning/association-rule-learning.html>
3. <https://www.hackerearth.com/blog/machine-learning/beginners-tutorial-apriori-algorithm-data-mining-r-implementation/>
4. http://www.salemmarafi.com/code/market-basket-analysis-with-r/