Towards Software that Predicts Food Preferences

Ghodratollah Aalipour
ga5481@rit.edu

Thomas Logan
tl9862@rit.edu

Shashank Rudroju
sr1632@rit.edu

Tejal Vishnoi
tv4466@rit.edu

ABSTRACT

In today's world, there is a seemingly endless number of food choices. People's preferences vary widely. For a person to accurately assess if she would like a food item, she may need to read all the ingredients and even understand their proportions and preparation methods. Additionally, she may also wish to gain confidence in her decisions by reading the reviews of like minded shoppers. Such effort is extremely time consuming and not practical given the vast variety of foods available. Thus, it is necessary that we develop techniques and tools to help people sort through the vast array of options and pick the foods that are most suitable for their palates. Towards this goal, we have begun to build software that can help users decide which foods to buy by mining data of food reviews and food ingredients.

1. OVERVIEW

Our project has two important components. The first is the data, which we use to gain useful insights into people's food interests. In section 2, we discuss the datasets we plan to use and some possibilities for extracting more data and finding patterns among various attributes of the data. The other important component is the software that will allow users to easily find out information related to their food interests. We discuss our software architecture and technology choices in section 3. Section 4 goes into more detail about how we will actually organize our data, and in section 5, we discuss how we transform our data to fit the schema in Postgres. We discuss our progress in building a web server in section 7 and we discuss approaches we are considering for our analysis in 8.

2. DATA

We will use the *Web Fine Food Dataset* [3], which contains data on food items sold by Amazon. The dataset contains information on 74,258 food products, 256,059 customers, and 568,454 reviews. The data consists of amazon IDs known as ASINs, customer IDs, customer profile names, review helpfulness ratings, product scores on a scale from 1.0 to 5.0, unix timestamps, summaries, and review texts. In addition to the attributes provided, we may wish to extract further attributes by parsing the natural language text, and also by looking up more information related to ASINs and customer IDs. For instance, we may look up the food ingredients for the provided ASINs either by using Amazon's public API[2], or extracting it from the HTML pages. We have organized our data into three related tables, for products, customers, and reviews using Postgres. After gathering further data from Amazon, we will populate our food table with more attributes, such as ingredients, prices and so on. We expect to find patterns in customers' food reviews and ratings that predict whether or not other customers will like various food products. The correlation of ratings and ingredients may also give indication of customers' dietary preferences.

3. SOFTWARE ARCHITECTURE

We use a three tier software stack to provide a means for users to learn more about their food preferences. The interface for querying the system and displaying results is being written in javaScript and will run as a web page within a web browser, such as Google Chrome. It will receive input from users and then make requests to our web server, which will be built using Node.js. The web server will interpret the request and in turn query our Postgres database server and calculate some result based on our statistical model. Our statistical model will be determined by writing programs in R to mine data in our datasets.

4. SCHEMA

We've created a Postgres database with three tables. The product table contains one attribute *asin* as the primary key; the customer table has attributes *id* as the primary key, and *name*. The review table contains attributes *product_asin, customer_id, time, score, summary, body* and *helpfulness*, where *product_asin* is a foreign key referencing the product table's *asin*, and *customer_id* is a foreign key referencing the customer table's *id*. We have not assigned a primary key for the review table since the data set has duplicate data for any of the attributes. The product table originally only had a single unique attribute that is also contained in the review table. However we have uncovered more attributes for product table, such as ingredients, editorials, labels, titles, prices, among many others.

To manage changes to our schema, we use sqitch [8]. Every time we wish to update our schema we simply add a new file with our DDL SQL. Sqitch keeps track of which

.

SQL statements have run and which have not. It alerts us if we try to apply changes that conflict with the current state of the database. Sqitch maintains three types of SQL files: *deploy*, *revert*, and *verify*, which enable us to gradually improve our schema, easily revert changes, or verify that we have correctly altered the state and schema of our database.

5. DATA MUNGING

Our initial data is formatted as a text file, where each review is separated by an empty line, and each attribute and corresponding value of a review is on its own line. We wrote javaScript to read in this data line by line, parse it, and produce a javaScript object for each review. The code uses *RxJs* [5] to enable overlapping of file IO with computation or other IO, such as generating SQL or inserting the data into Postgres.

To insert the data into Postgres, we wrote javaScript that calls our parsing function that returns an *Rx observable* object, which we can subscribe to, in order to receive reviews when available. We initially attempted to generate SQL and insert the data as soon as the review object was received, which was problematic. Each time a review object was received, we called the postgres server to insert the data. The next review object could be received before the previous review finished being inserted. By trying to insert as soon as each review object was available, we ran multiple insert queries concurrently, and observed consistency errors. Wrapping our SQL insertion statements in transactions did not solve our problem. Thus, we decided that each insertion must complete before the next. Additionally, we chose to buffer the parsed reviews and create a giant SQL statement that inserts up to 10,000 reviews at a time.

We modified our insertion code to place every parsed review in a list. Additionally we wrote a recursive insert function, to concurrently take up to 10,000 reviews from the list, generate an SQL insertion statement and make the insertion request to Postgres. Only after the insertion is successful, is the insert function called again to take more reviews from the list to process. With this technique, parsing and inserting the data into Postgres takes about four and a half to five minutes.

6. DATA FETCHING AND SCRAPING

Additionally, we also registered with Amazon to use their Product API, which gives us the ability to request further attributes about products or even perform searches for products. We have written code that looks up additional information for each product in our original data set. We plan to eventually integrate this information into our schema and database. Amazon has published an HTTP API, which requires some effort to use in javaScript. Fortunately, there already exists a javaScript wrapper [7], which we used instead of building our own. Using the product API, we discovered many attributes of products thay may provide useful insights in our analysis and application, such as the Amazon product page URL, product title, editorial review, price, image URL, brand, manufacturer, feature description, label, among others. Additional by scraping data from the product web page, we obtained ingredients, warning, and disclaimer data. We used cheeario [4] to parse the html page and extrat information for certain html elements. The part of the web page we scraped data from was marked with an HTML id called ”important-information”, containing multiple <div> elements, where each <div> element contains a <span> with a heading, such as ”Ingredients” and a <p> with some corresponding description. We extracted the data from the important information sections into javascript objects with the <span> text corresponding to object keys and the <p> text corresponding to object values.

7. WEB APPLICATION

To provide a means for users to take advantage of our organized data and eventual inference model, we have begun to build a web server to handle requests from users. We chose the library *Express*[1] to help us build our server. *Express* enables us to easily implement which HTTP requests we wish to respond to and how we wish to respond. We have built request handlers for the paths */products* and */customers* that make requests to the database and send their database responses back to the client as JSON.

Additionally, we have the server serve an HTML page for the root url path /. In the HTML we have placed links for both */products* and */customers*. Additionally we have added a bit of client side javaScript to respond to a button click by making an AJAX request to the server and updating the contents of the displayed HTML. We use *jQuery*[6] to help us make AJAX requests and update the client's UI.

We have begun the interface design for our food recommender software. The design consists of two pages. The first page displays a form prompting the user for information about her food preferences. Using this information our software will determine an initial list of food items to recommend. Once the user submits the form with her preferences, a second page will be displayed. The second page will display a horizontal list of ten or so food items. The user can indicate that she either likes or dislikes one of the items by clicking a ”thumbs up” or ”thumbs down” button. Based on the users selection the recommendations will be refined and new food item will be displayed replacing the decided item.

8. ANALYSIS TECHNIQUES

We plan to retrieve more attributes to complement our current data. For instance we will try to find an approximation of users' locations using timestamps of reviews. Then based on the location, it is likely that people living in the same time zone have the similar tastes for food. We will try to determine the highest selling brands and weight those heavily in our model function and system.

The algorithms we might use are KNN (K nearest neighbors), SVM (support vector machine) and the K-means algorithm. KNN is a clustering method that categorizes data based on the Euclidean distance function. Based on the category a person falls into, we recommend foods that other people in the same category find interesting. All three methods are classification algorithms where the first two are for supervised learning and the third is for unsupervised learning. We may also benefit from a good regression model. For instance, we can build a linear regression using the rating, the percentage of purchase of specific an item, the percentage of specific ingredients, the possible location, and other things. The classification approach may require many categories. A good model function and regression seems very plausible in this project. We may also use the brand along with rating to make recommendations. All of these depend on the way we ask users to the input information, which we are developing in correspondence with our analysis.

The code for KNN and K-means is available in several languages including R. From Coursera, we found an analogous version of our problem to recommend movies to users. It provides intuition about which attributes we should add. The general method for recommender systems is called collaborative filtering. But it is not the only method for building recommender systems. As we mentioned earlier, we can use a good logistic regression and use a threshold to make recommendations.

9. REFERENCES

[1] Express. 2016. Retrieved Oct 7, 2016, from `http://expressjs.com`.

[2] Amazon. Item lookup. August 2013. Retrieved Sep 9, 2016, from `https://docs.aws.amazon.com/AWSECommerceService/latest/DG/ItemLookup.html`.

[3] J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. *WWW*, 2013. Retrieved Sep 9, 2016, from `http://snap.stanford.edu/data/web-FineFoods.html`.

[4] M. Mueller. Cheerio. 2016. Retrieved Nov 4, 2016, from `https://github.com/cheeriojs/cheerio`.

[5] M. Podwysocki. Rxjs. 2016. Retrieved Oct 7, 2016, from `https://github.com/Reactive-Extensions/RxJS`.

[6] J. Resig. jquery. 2016. Retrieved Oct 7, 2016, from `http://jquery.com`.

[7] S. Thiboutôt. Amazon product advertising api client. 2016. Retrieved Oct 7, 2016, from `https://github.com/t3chnoboy/amazon-product-api`.

[8] D. E. Wheeler. Sqitch. 2016. Retrieved Oct 7, 2016, from `http://sqitch.org`.