Review-Based Product Recommendation Using Latent Semantic Analysis

Midterm Project Update

Ben Nissan and Cuong Nguyen

April 3, 2018

1 Problem Statement

Online commerce, or e-commerce, has become a universal force across countless industries over the past 20 years, hallmarked by a major advantage over brick-and-mortar stores: online reviews. Happy customers leave positive reviews while dissatisfied customers leave negative reviews, and crucially, these reviews can be acted upon immediately by other customers. Product recommendation systems have thus become central tools for major online stores, such as Amazon. We argue that online reviews constitute powerful enough data sets to build a robust review-based product recommendation system—a system useful enough to recommend new and alternative products for happy and unhappy customers alike.

2 Approach

We plan to approach this problem using a latent semantic analysis (LSA) model. We will initially work to train the model to differentiate products based on broad categories. For example, after training the model on many shoe reviews, it should be able to match future reviews about shoes. Once the model has been trained on features for various product categories, we can pass an arbitrary review into it and categorize the review accordingly.

To translate this into explicit product recommendation, we will focus on recommendations within existing categories. After a category has been identified, we can take the set of reviews within it and use it as our recommendation pool. Upon matching a new review, we can recommend products in the closest category to its match. Similarities between reviews will be determined by distance criteria in review feature space; our first approach for a distance criteria will be the cosine distance, but this may not be our final method.

Because LSA is susceptible to polysemy (where one word has many meanings) and synonymy (where multiple words have the same meaning), we will consider adding conditional probabilistic analysis to gain more contextual interpretations of customer reviews.

3 Evaluation

We will separate our evaluation process into two phases. First, we will test the feature identification abilities of our LSA model. By training our model solely on a single product category of reviews, we can gauge its efficacy by testing it on both products of the same category and products of different categories; ideally, it should present bias towards products of the same category. Initially, we will train our model only on a subset of Amazon reviews for shoes and movies. After training, if we pass in a movie review, the model should score the review high with respect to movie features and low with respect to shoe features. From there, we plan to build to larger and more diverse pools of product categories and sources, expanding even beyond e-commerce to service industry reviews.

For the second phase of our evaluation process, we will test how well our recommendation system works. This phase ultimately comes down to subjective evaluation, but is nontrivial for any real-world recommendation system. To maintain a level of objectivity, our system will be evaluated based on its ability to match multiple reviews for the same product, as well as other similar products. For example, we would expect our system to rank highly reviews for various Marvel movies if asked what reviews resemble those for Black Panther. Ideally, our system should recommend multiple of such films, even though Black Panther itself should have the highest recommendation score.

4 Materials

4.1 Data

We plan to use data from the following sources to train our model:

- Amazon Fine Food Reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)
- Amazon Product Data (http://jmcauley.ucsd.edu/data/amazon/)
- Amazon Reviews (https://snap.stanford.edu/data/web-Amazon.html)
- Yelp Open Dataset (https://www.yelp.com/dataset)

4.2 Code Libraries

We plan to use code from the following libraries in our project:

- Breeze
- Play JSON

5 Results

We focused the first stage of our development on obtaining, parsing, and processing our training data sets. Our first goal was to choose among the data sets we found for our initial project proposal (listed above under Data) to find a collection well-suited to the problem of training an LSA model. After discussion and consideration, we settled on the Amazon Product Data sets for a number of reasons.

Primarily, these data sets offer us benefits of scale and data diversity; they contain 142.8 million reviews separated into 24 diverse product categories. Furthermore, they offer 10 million of those reviews similarly categorized in the form of 5-core data sets, where each item and user has a total of five reviews, all guaranteed to be unique. Finally, they offer a wide breadth of review counts by category: the smallest category set (musical instruments) has roughly 10,000 reviews and is useful for testing our data processing code as we develop, while the largest (books) has nearly 9 million reviews and will be more useful later on when training the model itself.

After selecting the Amazon Product Data sets, we proceeded to the challenge of properly parsing it from JSON into a term-count table useful for training an LSA model. We model this table as a map from terms—unique word strings found in a given data set—to integer counts of how often they occur in the data set. To parse a given data set, we first read it in as JSON line by line; as each review is stored on a separate line, this inputs the data set one review at a time. Then, we extract the text of the review, stripping punctuation characters, and split it into an array of words. Finally, we update the count of each word in the review in the map, adding new words as needed. The `DataMap` class also offers functions to extract the terms and counts separately, as arrays of strings and integers, respectively.

Finally, we began our implementation of the LSA itself. We first use our `DataParser` to parse an input data set into its associated data map. We then use the provided `getCounts()` function to pull the term counts, and convert the resulting array into a `DenseVector`. By taking the product of the vector and its transpose, we obtain a `DenseMatrix` corresponding to a correlation matrix for the term counts. Our next challenge is to use the eigenvalues and eigenvectors of this matrix to weigh each individual term's contribution to its topic.

To run our data processing code, open `sbt` in the `productrec` project folder and call `run src/resources/Musical_Instruments_5.json` (optionally replacing the filepath with another Amazon Product Data file of your choice). This will process the data into its associated map, print the map, and (for ease of interpretation given the large size of the map), print the size of the map.