Review-Based Product Recommendation Using Latent Semantic Analysis

Pre-Final Project Update

Ben Nissan and Cuong Nguyen

May 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 are using data from the following sources to train and test our model:

- Amazon Product Data (http://jmcauley.ucsd.edu/data/amazon/)

4.2 Code Libraries

We are using 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-document matrix useful for training an LSA model. We provide two separate versions of this matrix: one that stores the term frequency of each term in each document, and one that stores the term frequency-inverse document frequency. The internal representation of the matrix uses the TF version. 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 matrix, adding new words as needed.

To convert to the TF-IDF version, we first sum the number of non-zero values across each row (i.e., each document) of the TF matrix, divide the resulting reduced vector by the total number of documents, and map across it take the negative log of each element. We then convert the vector to a scaling matrix using Breeze's `diag()` function, and multiply it by the original TF matrix to obtain our final TF-IDF matrix. The `TermDocumentMatrix` class offers functions to extract both versions of the matrix separately—although we have found that the TF-IDF version produces more compelling LSA results—as well as one to extract an array of all terms.

Finally, we implemented the LSA algorithm itself. We first use our `DataParser` to parse an input data set into its associated term-document matrix: in our use case, a TF-IDF matrix. Given that matrix, we run a fast, truncated SVD algorithm to extract two matrices,`U` and `Vt`, and one vector, `S`. Given a constant, `k`, that is provided to the `fastTruncatedSVD()` function, the columns of `U` represent the importance of each term to each one of `k` topics, while the rows of `V` represent the importance of each document to each topic. We chose an appropriate `k` by first selecting an arbitrary value, then plotting the resulting elements of `S` and choosing a new value based on the inflection point of the resulting curve; from our subjective perspective, 10-20% of the total number of documents tends to be a good choice.

To interpret `U`, we select an "importance threshold" for each column equal to the 99th percentile of positive values in the column, filter out the indices of all elements in each column greater than or equal to that threshold, and select the corresponding terms to obtain the terms most relevant to each topic. Similarly, to interpret `V`, we select an threshold for each row and filter out the indices of all elements in each column greater than or equal to that threshold; the indices themselves correspond to the most relevant documents.

This algorithm has provided us with promising results, even on relatively small data sets. Using a 500-review training data set with a `k`-value of 50, we found that `U` contained many topics of grouped words with similar meanings. One topic included the words "string", "guitar", "bridge" and "neck", appearing to focus on guitar components. Another included the words "processor", "audio", "microphone" and "board", appearing to focus on recording equipment. These early results suggest that our LSA implementation already provides some meaningful term clustering.

In our final submission, we plan to use `V` to select between more and less meaningful documents. Given this information, we can filter out irrelevant or generic reviews before running LSA on the resulting subset of data, allowing us to pre-select for more contrast between topics. In addition, we intend to develop a means of applying our trained LSA model to the problem we set out to address initially: product recommendation. This will primarily rest on a method of associating new reviews with our obtained topics: our current idea is to do so based on shared word frequency.

To run our program, open `sbt` in the `productrec` project folder and call `run src/resources/Musical_Instruments_5_500.json 50` (optionally replacing the filepath with another Amazon Product Data file of your choice or replacing the second argument (`k`) with any positive value less than or equal to the number of reviews in the file. This command will generate a `TermDocumentMatrix`, run `LSA`'s `getDocumentTopics()` and `getTermTopics()`, and print the resulting topic lists to the terminal.