Sayer Rippey, Oren Shoham, Peter Fogg
CS364
Final Report

Introduction

Problem

System Design

The system is broken up into three parts – data collection, data processing, and the core recommendation engine.

- **Collection.** 904,149 reviews were scraped from BeerAdvocate[1], using the BeautifulSoup library to parse HTML. The text of each individual review was saved to a SQLite database, along with the score, author's username, and the associated beer's name, brewery of origin, and percent ABV.
- **Processing.** After obtaining the data, we examined the 400 most commonly used words across all reviews. From these, a set of 85 particularly descriptive words was hand-selected. Our hypothesis was that words like "grapefruit", "hoppy", or "toasted" would appear frequently in reviews of beers with those attributes, and that similar beers would have similar frequencies of attribute usage in reviews. For each attribute, we summed the number of appearances of the attribute in all reviews of a particular beer and normalized by the number of reviews. Each of the 9442 beers is then represented as a vector of attributes in the range $[0, 1]$.
- **Recommendation.** We use a k-nearest neighbors algorithm to determine similar beers. Given an input beer and a k, the beer's vector representation is retrieved from the database, and all other beers are compared against it using a simple Manhattan distance function. The nearest k beers are then printed out, sorted in increasing order of distance from the input.

  There are a few additional features for the user's convenience. In order to give more guidance as to which recommended beers the user will enjoy, we print out the most similar between the input beer and a recommendation – for example, searching for Sierra Nevada Torpedo tells me that Hop Notch is similar in the attributes "bite", "apple", and "foamy". We also tell the user the most important attributes of a particular beer, which are assumed to be those which appear most commonly in reviews. Torpedo's most important attributes are "hop", "pine", and "red".

  Beer names are somewhat inconsistently formatted in our database, and the user might not remember the precise name of the beer they want. For example, the user might search

[1] http://beeradvocate.com

for “Torpedo”, but the database contains “Sierra Nevada Torpedo Extra IPA”. As such, we implemented a fuzzy string matching heuristic to suggest similarly-named beers. If the input is a substring of a beer name, this distance heuristic is 0. Otherwise, it is the length of the longest common subsequence plus 3 over the Levenshtein distance of the two strings. We return a sorted list of beernames with a sufficiently small distance, and present these to the user as options. The heuristic is somewhat arbitrary, but it usually suggests the desired beer as the first result.

Evaluation

Given that beer preferences are an entirely subjective matter, we used a qualitative approach to our testing. When prompted with a beer that the user likes, the system should respond with similar beers. Better yet, the results should contain other beers that the user likes – this is a good indication that the system is capable of determining the user’s preferences.

Our results are promising. The system always returns beers of a similar style and quality, and often gives beers that the testers are known to enjoy.

Related Work

Future Work

Conclusion