Sayer Rippey, Oren Shoham, Peter Fogg
CS364
Final Report

---

## Introduction

How do you know that you'll like a beer before you even try it? Machine learning! The goal of our project was to create a machine learning-based recommender system along the lines of Pandora or Netflix, but for beers. In other words, we wanted to make a program that would take in an input beer and return a list of similar beers that the user might like. A beer, like a piece of music or a movie, requires a complex description that incorporates a number of different attributes. In order to measure similarity between beers, we first needed to find a way to quantify these attributes. We obviously didn't have time to create a quantitative description of every beer in the world by hand, à la Pandora. However, we did have one thing — the Internet! Specifically, beer review websites. We scraped a bunch of reviews from a popular site and wrote a relatively simple NLP progam to generate attribute vectors for each beer.

There are two main approaches to designing a recommender system: content-based filtering and collaborative filtering. Content-based filtering algorithms compare items based on the characteristics of those items. Collaborative filtering algorithms, on the other hand, compare items based on the users who like or dislike them (e.g. you like $x$, other people who like $x$ also like $y$, so you might like $y$). For our recommender system, we decided to try several approaches: a content-based algorithm, a collaborative algorithm, and a hybrid of the two.

## Problem Definition and Algorithms

Formally, our problem can be formulated as:

**Input:** a single beer, supplied by a user

**Goal:** a list of similar beers that the user might also like

Each of our three algorithms is a variation on the $k$-nearest neighbors algorithm. Our content-based algorithm takes in an input beer and returns the $k$ beers whose attribute vectors are closest to that of the input beer in terms of Manhattan distance. Our collaborative algorithm takes in an input beer, compiles a list of beers that users who liked the input beer also liked (we assumed that a user "liked" a beer if they gave it a score of 4.0 out of 5.0 or higher), and returns the $k$ most highly rated beers on that list. Our hybrid algorithm takes in an input beer, compiles a list of beers that users who liked the input beer also liked, and returns the $k$ beers from that list whose attribute vectors are most similar to that of the input beer in terms of Manhattan distance.

## 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, we hand-selected a set of 85 particularly descriptive words. 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 their 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 for that beer. Each of the 9442 beers is represented as a vector of these attributes, with each attribute value 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 attributes (i.e. those with the most similar non-zero values) 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 (i.e. the attributes with the highest values). 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 for "Torpedo", but the database contains "Sierra Nevada Torpedo Extra IPA". In order to compensate for this, 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.

[1] http://beeradvocate.com

## 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.

## Future Work

## Conclusion