Craft Beer Recommendation Engine
Shelby Heise

Problem Statement

Ready to find your next craft beer?

As a craft beer drinker, I wanted to build a basic recommendation engine to help others find their next favorite beer based on beer they already enjoy. This project is an example of content-based recommendations.

Data Collection

Data for this project was pulled from Kaggle, and can be found here . In January 2017, Kaggle user Jean-NicholasHould scraped the now defunct CraftCans.com to create a dataset containing information on 2000+ canned craft beer varieties and 500+ US breweries.

A tutorial for their web-scraping method can be found here.

The basis for this project draws heavily from the datacamp course "Building Recommendation Engines in Python."

Data Cleaning

As the data for this project was already in good shape, there wasn't much initial cleaning required. The csv files containing brewery information and beer information were merged on brewery_id, and a few column names were changed for clarity. Additional columns were dropped due to missing values. The final dataframe consisted of columns containing information on the beer name, brewery name, alcohol by volume (abv), and brewery location (city).

Jaccard Similarity – Processing

For this project, two content-based recommendation engines were created. The first focused on Jaccard similarity, which is measured by comparing the number of shared attributes of two items divided by the total number of combined attributes. The Jaccard similarity score ranges from 0 to 1, with 1 being a "match/most similar." In this project, the Jaccard similarity was measured between each beer based on beer_style (lager, IPA, etc.).

The first step in this process was to apply pandas' crosstab function to the merged and cleaned dataset to create a vectorized version of the beer attributes. To calculate the Jaccard similarity scores for the entire dataset, scipy's pdist was applied to the values of the vectorized attributes with the calculation metric set as "jaccard." Pdist actually measures the distances (differences) between the beers rather than the similarity. To create a similarity array, the results were subtracted from 1 and then converted to a dataframe.

The resulting dataframe is filtered based on the preferred beer to produce several beer recommendations.

```
#Testing recommendations for an OKC craft beer
arjuna_similarity = beer_style_similarity_df['Arjuna']

arjuna_similarity = pd.DataFrame(arjuna_similarity.sort_values(ascending = False))
```

```
#15 recommendations for Arjuna
arjuna_similarity.head(15)
```

	Arjuna
beer_name	
Great Crescent Belgian Style Wit	1.0
Upland Wheat Ale	1.0
Arjuna	1.0
Pinata Protest	1.0
Plum Island Belgian White	1.0
Cold Snap	1.0
Lost Meridian Wit	1.0
Cotton Mouth	1.0
What the Butler Saw	1.0
White (2015)	1.0
White Ale	1.0
White Magick of the Sun	1.0
White Rabbit	1.0
White Rascal	1.0
White Thai	1.0

As you can see in the screenshot, all of the recommended beers for Arjuna share the beer type “Witbier.” As this model only considers beer type, it is not able to offer very complex recommendations.

Cosine Similarity – Processing

The second recommendation engine for this project focused on cosine similarity. This metric “quantifies the similarity between two or more vectors” (Alake 2020). For this project, cosine similarity was measured on text-based similarities between each beer. Cosine similarity scores measure from 0 to 1, with 1 being an exact match.

Craft Beer Recommendation Engine
Shelby Heise

This method is dependent on analyzing text-based similarities and requires an item "description" rather than the clearly labeled attributes (abv, beer_style, etc.). As the dataset did not contain a "description" column, the dataframe was converted to string and the abv, beer_style, brewery_name, and city columns were concatenated to create a description for each beer.

TF-IDF—term frequency inverse document frequency—is used to convert the item description into something that can be used for comparison. TFIDF separates out each word into a feature, and gives greater weight to the most "important" words to filter out commonly used words (like "the"). The TfidfVectorizer from scikit-learn handily completes this step and was applied to the dataframe containing just the column for beer name and the description column. The resulting vectorized data can now be used to determine cosine similarity. Calculating cosine similarity for the entire dataframe can be easily completed by using scikit-learn's cosine_similarity function on the vectorized dataframe.

Like with the Jaccard example, the final dataframe can be filtered based on the preferred beer to generate recommendations. For this example, a field for user input was added to create a basic recommendation engine.

```
#Testing for Arjuna, local OKC beer
cosine_sim_series = cosine_sim_df.loc['Arjuna']

beers_simliar_to_Arjuna = cosine_sim_series.sort_values(ascending= False)
```

```
beers_simliar_to_Arjuna.head(10)
```

```
beer_name
Arjuna                          1.000000
Uroboros                        0.775140
Golden One                      0.765317
F5 IPA                          0.466419
Gran Sport                      0.458117
Horny Toad Cerveza (2013)       0.445524
Horny Toad Cerveza              0.445524
Native Amber (2013)             0.432591
Native Amber                    0.432591
RoughTail IPA                   0.291590
Name: Arjuna, dtype: float64
```

In this screenshot, you can see the recommended beers are very different from the Jaccard model. This model considers more than just beer type, and can offer a more complex recommendation.

Final Recommendation Engine

Craft Beer Recommendation Engine
Shelby Heise

The final recommendation engine is based on cosine similarity, as it allows for a more "nuanced" beer recommendation than the Jaccard score. A dropdown list containing all of the available beers for selection was also added using ipywidgets. The dropdown ensures that the user is only able to select a beer that exists in the dataset and omits possible spelling/capitalization errors.

While the Jupyter notebook in Github is "static," the required data for replication has been added to this repository.

Takeaways

This project was a great first step in building a recommendation engine.

By far, the biggest limitation was the dataset. The "description" column created for this project consisted of just the brewery name, brewing location, beer style, and abv. As a result, the recommendations were based on only a few factors. With information on beer flavors, brewing methods, distribution areas, etc., it would be possible to create a much more robust recommendation engine. Additionally, the dataset only contains information on *canned* craft beers, which leaves out any beers that are only available in bottles. Furthermore, the data was scraped in 2017. Included breweries may no longer have the same set of beers available or may even be closed. Finally, the website used to collect the dataset is no longer around. As a result, it is not possible to personally verify the accuracy of the scraped data.

In future versions of this project, I would also like to roll out a web app alongside the recommendation engine.

Tools & Documentation

For this project, the following tools and sources were used:

- scikit-learn
  - Associated Documentation – Jaccard
  - Associated Documentation – TF-IDF
  - Associated Documentation – Cosine
- scipy
  - Associated Documentation

- ipywidgets
  - Associated Documentation

- Alake, Richmond. "Understanding Cosine Similarity And Its Application." *Towards Data Science*, 14 September 2020. towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a .

Craft Beer Recommendation Engine
Shelby Heise

- Jean-NicholasHould. “Craft Beers Dataset.” *Kaggle.* kaggle.com/nickhould/craft-cans?select=beers.csv .

- O’Callaghan, Rob. “Building Recommendation Engines in Python.” *Datacamp.* datacamp.com/instructors/mrrocallaghan .