Brewing Up Data: Using Web Scraping to Make Novel Beer Recommendations

Sam O'Neill, Aaron Plesset, Xioaxiong Xu, and Xiaoyue Zhu

University of California, Santa Barbara

June 8, 2018

Overview

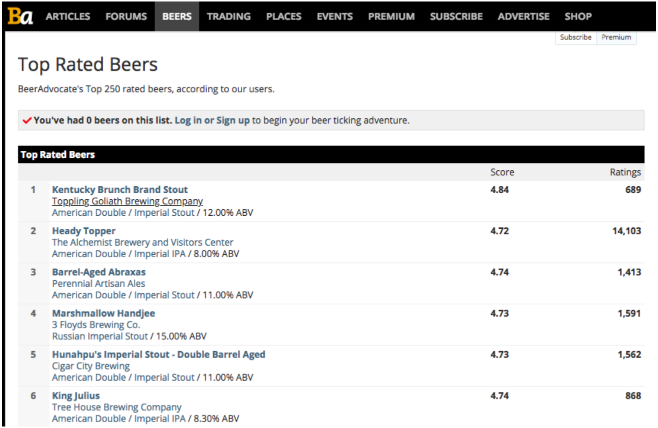
1. Introduction
2. Data Engineering
   - Method 1: wget and grep
   - Method 2: custom HTML parsing
3. Data Analysis
   - Data Visualization
   - Recommendation Systems
4. Conclusion/Future Work

Brief introduction

BeerAdvocate.com is essentially Yelp for beers. The website has thousands of users who typically review hundreds of beers. Thus, there is a plethora of data stored in this website that is ripe for use. However, there is no public API which makes data extraction somewhat of a nightmare.

In this project we attempt to solve this extraction problem by creating our own API with two different web scraping methods. With the resulting structured data, we perform exploratory data analysis and create model recommendation system.

A quick look at BeerAdvocate.com



Ba ARTICLES FORUMS BEERS TRADING PLACES EVENTS PREMIUM SUBSCRIBE ADVERTISE SHOP

Subscribe Premium

Top Rated Beers

BeerAdvocate's Top 250 rated beers, according to our users.

✔ **You've had 0 beers on this list. Log in or Sign up** to begin your beer ticking adventure.

Top Rated Beers

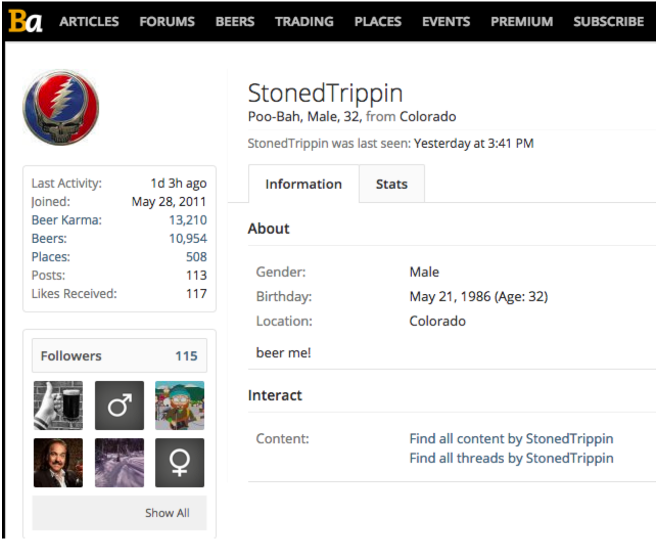
		Score	Ratings
1	**Kentucky Brunch Brand Stout** Toppling Goliath Brewing Company American Double / Imperial Stout / 12.00% ABV	**4.84**	**689**
2	**Heady Topper** The Alchemist Brewery and Visitors Center American Double / Imperial IPA / 8.00% ABV	**4.72**	**14,103**
3	**Barrel-Aged Abraxas** Perennial Artisan Ales American Double / Imperial Stout / 11.00% ABV	**4.74**	**1,413**
4	**Marshmallow Handjee** 3 Floyds Brewing Co. Russian Imperial Stout / 15.00% ABV	**4.73**	**1,591**
5	**Hunahpu's Imperial Stout - Double Barrel Aged** Cigar City Brewing American Double / Imperial Stout / 11.00% ABV	**4.73**	**1,562**
6	**King Julius** Tree House Brewing Company American Double / Imperial IPA / 8.30% ABV	**4.74**	**868**

A quick look at BeerAdvocate.com

The HTML under the hood:

```
<div style="font-size:1em;">
<div id="rating_fullview"><div id="rating_fullview_container" class="user-comme
background:#E8E8E8;"><a href="/community/members/jmonah3.887293/" class="userna
width="48" height="48"  border="0" alt="Photo of Jmonah3"></a></div></div><div
class="rAvg_norm">/5</span>  rDev <span style="color:#006600;">+3.3%<
5</span><br><br><div><span class="muted"><a href="/community/members/jmonah3.88
ba=Jmonah3#review">Sunday at 09:19 PM</a></span></div></div></div><div id="rati
id="rating_fullview_user"><div style="padding:3px; background:#E8E8E8;"><a href
src="styles/default/xenforo/avatars/avatar_male_s.png" width="48" height="48"
<span class="BAscore_norm">5</span><span class="rAvg_norm">/5</span>  
5 | taste: 5 | feel: 5 |  overall: 5</span><br><br><div><span class="muted"><a
href="/beer/profile/23222/78820/?ba=Bugsmcl#review">May 22, 2018</a></span></di
user="957865"><div id="rating_fullview_user"><div style="padding:3px; backgroun
src="https://cdn.beeradvocate.com/data/avatars/s/957/957865.jpg?1440623969" wid
id="rating_fullview_content_2"><span class="BAscore_norm">4.85</span><span clas
<span class="muted">look: 5 | smell: 5 | taste: 4.75 | feel: 4.5 |  overall: 5<
class="username">Aerizel</a>, <a href="/beer/profile/23222/78820/?ba=Aerizel#re
```

A quick look at BeerAdvocate.com



Web scraping via wget and grep

We first scraped data with wget and grep to form a Beer class and User class. The Beer class stores the stats, info, and ratings of a particular beer. The User class stores similar data but for a specific user.

Below is the scrape_burger() function that we used in many different ways to create these classes.

```
def scrape_burger(url = '', top_bun = '', bottom_bun = '', patty = '', napkins=False):
    """
    Scrapes a website's html for text located between the 'top bun' (left side) and 'bottom bun' (right side).
    Patty argument is for fine tuning the scraping using regex. Lastly, napkins are for cleaning up the mess.
    """
    url = '\''+ url + '\''
    bottom_bun = ')\'' if bottom_bun=='' else '?)(?=' + bottom_bun + ')\''
    burger = '\'(?<=' + top_bun + ')' + patty + '(.*'+ bottom_bun
    if napkins:
        cleanup = '\'(?<=' + napkins + ')' + '(.*)\''
        x = !wget -qO - {url} | grep -oP $burger | grep -oP $cleanup
    else:
        x= !wget -qO - {url} | grep -oP $burger
    #if regex search fails return space character
    if len(x)==0:
        return " "
    return x
```

Example of scraped data from method 1

```
test_user=User(URL1)
```

```
test_user.info
```

```
{'Gender': 'Male',
 'Birthday': 'May 21, 1986 (Age: 32)',
 'Location': 'Colorado'}
```

```
test_user.stats
```

```
{'BeerKarma': '13,210',
 'NumRatings': '10,954',
 'NumPosts': '113',
 'NumLikes': '117'}
```

```
test_dict=test_user.get_ratings(50)
for k in range(0,5):
    print(test_dict['beers'][k],test_dict['ratings'][k])
```

```
Budweiser Freedom Reserve Red Lager 3.42
Bud Light Orange 2.37
Si Cerveza 4
Jolly Pumpkin / Stillwater - Losing Our Ledges 4.25
Frond 3.6
```

Pros and cons of method 1

Pros:

- Able to scrape various pieces of information for both beers and users
- Successfully extracts data for all public users tested

Cons:

- Does not extract all relevant data for beers and users
- Tedious to manually search for all pieces of information in HTML text
- Somewhat inefficient as we call wget on the same web page multiple times (It would be quicker to parse the entire web page at once for data)

Web scraping via custom HTML parsing

In order to address the issues from the first method, we tried a different web scraping approach. This approach involved using an HTML parser to scrape all useful information in one go instead of calling wget several times. This became our method of choice and was developed into a python package called Beer_Advocate_API.

Package Download: `pip install Beer_Advocate_API`
Documentation: In development

Example HTML code

```
    <div id="info_box" style="float:right;width:70%;" class="break">

        <div id="main_pic_norm" style="text-align:center; float:right; width:150px; padding:0px;
src="https://cdn.beeradvocate.com/im/beers/78820.jpg" width="150" height="300" border="0"
alt="&#75;&#101;&#110;&#116;&#117;&#99;&#107;&#121;&#32;&#66;&#114;&#117;&#110;&#99;&#104;&#32;&
style="position:absolute;left:0;top:0;"><img src="https://cdn.beeradvocate.com/im/c_beer_image.g
alt="&#75;&#101;&#110;&#116;&#117;&#99;&#107;&#121;&#32;&#66;&#114;&#117;&#110;&#99;&#104;&#32;&
</div></div>

        <b>BEER INFO</b>

        <br><br>
        <b>Brewed by:</b>
        <br>
        <a href="/beer/profile/23222/"><b>Toppling Goliath Brewing Company</b></a>
        <br><a href="/place/directory/9/US/IA/">Iowa</a>, <a href="/place/directory/9/US/">Unite
target="_blank">tgbrews.com</a>     <br><br>
        <b>Style:</b> <a href="/beer/style/157/"><b>American Double / Imperial Stout</b></a>
        <br><br>
        <b>Alcohol by volume (ABV):</b> 12.00%
        <br><br>
        <b>Availability:</b> Rotating
        <br><br>
        <b>Notes / Commercial Description:</b>
        <br>
        This beer is the real McCoy. Barrel aged and crammed with coffee, none other will stand
difficult to track down. If you can find one, shoot to kill, because it is definitely wanted...
</div>
```

Example of scraped data from method 2

```
test = Beer('/beer/profile/23222/78820/')
test.info
```


```
{'Brewed by': ['Toppling Goliath Brewing Company',
  'Iowa',
  'United States',
  'tgbrews.com'],
 'Style': ['American Double  Imperial Stout'],
 'Alcohol by volume (ABV)': [' 12.00%'],
 'Availability': [' Rotating'],
 'Notes  Commercial Description': ['This beer is the real McCoy. Barrel aged and crammed with
stand in it’s way. Sought out for being delicious it is notoriously difficult to track down. :
t to kill because it is definitely wanted... dead or alive.',
  'Added by siradmiralnelson on 02-26-2012'],
 'Ranking': ['#1'],
 'Reviews': ['132'],
 'Ratings': ['689'],
 'Bros Score': ['0'],
 'Wants': ['3701'],
 'Gots': ['103'],
 'Trade': ['5']}
```

Pros and cons of method 2

Pros:

- Able to scrape all relevant info at once
- Able to scrape more info
- Easier to specify which information should be scraped on HTML code

Cons:

- Still somewhat computationally intensive when scraping lots of users/beers
- Possible bugs as we have only had a week to test it (although no noticeable ones have come up)

Final beer class

Important attributes:

- info : (dict) Dictionary of beer info and stats from the beer's profile page
- main_html : (str) HTML for beer's profile page

Functions:

- get_name(): returns beer name
- get_reviews(): returns dictionary of specified number of beer reviews on beer's page

Final user class

Important attributes:

- user_id : (str) The user's user_id if the user's profile page is public
- info : (dict) Dictionary of user info from the user's public profile page
- reviews: (str) Contains first page of user's reviews

Functions:

- get_reviews(): returns list of specified number of beer reviews on user's page

Creating structured dataframes

With the User and Beer classes we can now somewhat easily extract lots of relevant data. For example we perform the following steps when creating the ratings matrix that is used for our recommendation systems:

- Scrape list of usernames from beer page
- For each username in list, scrape a specified number of beer reviews from his/her user page
- Find most-rated beers from scraped user reviews
- Create a ratings matrix using the *n* most-rated beers. This matrix can be tuned according to a sparsity constraint so users with little to no reviews are excluded.

Creating structured dataframes

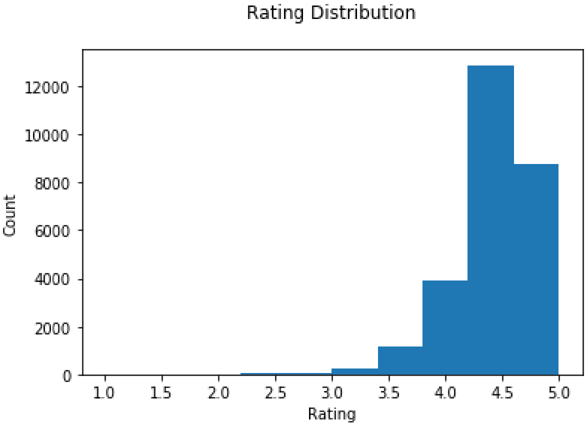
	Bugsmcl	Humbolt9	Jimmeekrek	StonedTrippin	jsearley3364	Chadlossie	WOLFGANG	fossage78
Kentucky Brunch Brand Stout	5.00	5.0	4.81	0.0	4.91	5.0	5.0	5.00
Heady Topper	0.00	0.0	0.00	0.0	4.81	0.0	0.0	0.00
Mornin' Delight	4.56	0.0	4.63	0.0	4.91	0.0	0.0	0.00
Hunahpu's Imperial Stout - Double Barrel Aged	0.00	0.0	0.00	0.0	0.00	4.0	0.0	0.00
Barrel-Aged Abraxas	4.91	0.0	5.00	0.0	0.00	5.0	0.0	4.25

Creating structured dataframes

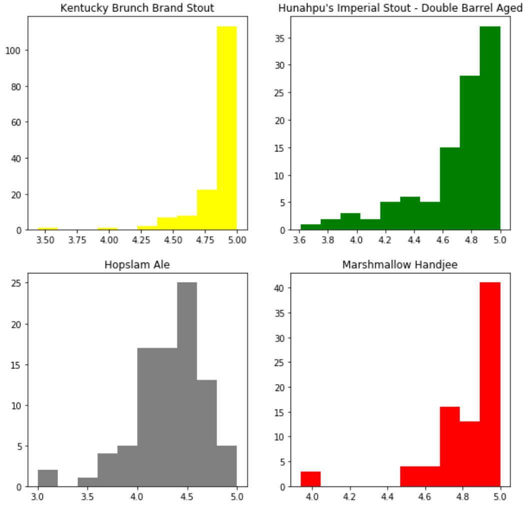
	brewery	state	country	website	style	abv	availability	description	ranking
Fat Tire Amber Ale	New Belgium Brewing	Colorado	United States	newbelgium.com	American Amber Red Ale	5.20%	Year-round	No notes at this time.	#40195
Nugget Nectar	Tröegs Brewing Company	Pennsylvania	United States	troegs.com	American Amber Red Ale	7.50%	Spring	Squeeze those hops for all they're worth! Nugg...	#451
Hop Head Red Ale	Green Flash Brewing Co.	California	United States	greenflashbrew.com	American Amber Red Ale	8.10%	Year-round	In 2011 the recipe was altered to bump the IBU...	#5041
Amber Ale	Bell's Brewery - Eccentric Café & General Store	Michigan	United States	bellsbeer.com	American Amber Red Ale	5.80%	Year-round	The beer that helped build our brewery; Bell's...	#12977
Hopback Amber Ale	Tröegs Brewing Company	Pennsylvania	United States	troegs.com	American Amber Red Ale	6.00%	Year-round	Standing 12 ft. tall at the center of the brew...	#5046

Observing data

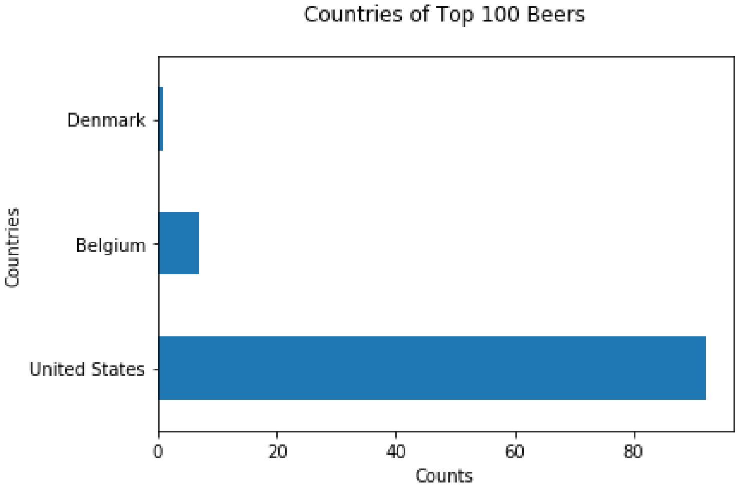
600 kinds of beers rated by 159 users



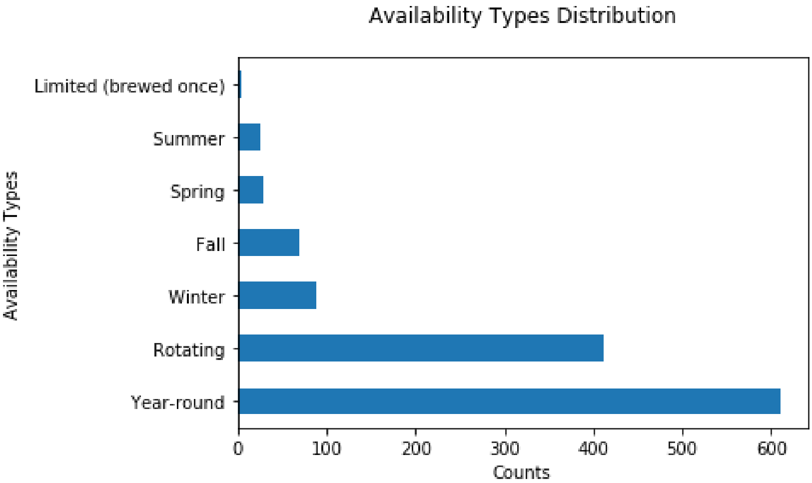
Observing data



Observing data



Observing data



Recommendation Systems

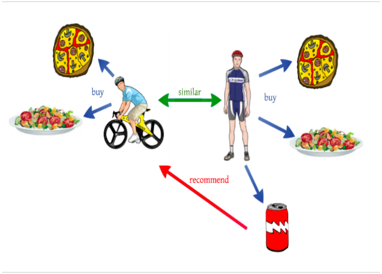
- Content-based algorithms (X)
- Collaborative filtering algorithms
  - Memory-based collaborative filtering
    - User-to-User CF
    - Item-to-Item CF
  - Model-based collaborative filtering

Why collaborative filtering?

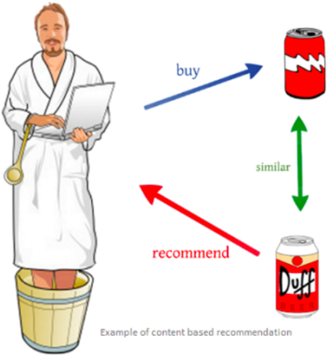
- Personalization (compared to Popularity Recommendation)
- Ability to handle large data compared to Classifier Recommendation
- Flexibility to cross different domains

Memory-based Collaborative filtering recommendations

Goal: predict how well a user will like an item that he has not rated



(a) User-to-User

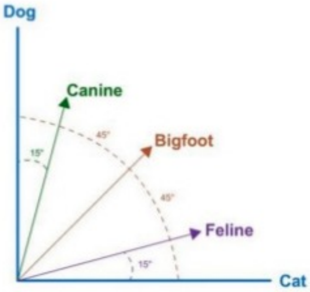


(b) Item-to-Item

Figure: Types of collaborative filtering recommendation

Cosine similarity

- Viewing two items(or users) and their rating as vectors, define the similarity between them as the angle between these vectors.



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

User-to-User CF

Cosine Similarity Matrix:

```
cosine_sim = 1-pairwise_distances(data, metric="cosine")
pd.DataFrame(cosine_sim)
```

	0	1	2	3	4	5	6	7	8
0	1.000000	0.537896	0.114253	0.185072	0.605874	0.176214	0.525800	0.208713	0.310875
1	0.537896	1.000000	0.153032	0.170818	0.455301	0.243024	0.490277	0.264025	0.204986
2	0.114253	0.153032	1.000000	0.179889	0.128106	0.137525	0.156140	0.143291	0.096687
3	0.185072	0.170818	0.179889	1.000000	0.256825	0.100531	0.081392	0.200322	0.447156
4	0.605874	0.455301	0.128106	0.256825	1.000000	0.219340	0.411796	0.236726	0.322890
5	0.176214	0.243024	0.137525	0.100531	0.219340	1.000000	0.245627	0.231765	0.151232
6	0.525800	0.490277	0.156140	0.081392	0.411796	0.245627	1.000000	0.266766	0.197514
7	0.208713	0.264025	0.143291	0.200322	0.236726	0.231765	0.266766	1.000000	0.182233
8	0.310875	0.204986	0.096687	0.447156	0.322890	0.151232	0.197514	0.182233	1.000000
9	0.146033	0.224688	0.069247	0.216337	0.189133	0.155449	0.096326	0.221928	0.167699
10	0.539385	0.645535	0.108554	0.135855	0.455201	0.241060	0.567100	0.266852	0.282161

Making prediction (User-based)

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in K} (r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}}$$

- $P(a,i)$: the prediction rating for user a to item i
- $\bar{r}_a$: the average rating of user a to all items
- $r_{u,i}$: is the rating user u gives to item i
- $\bar{r}_u$: is the average rating of user u to all items
- $r_{u,i} - \bar{r}_u$: the difference between the rating of an user give item i and this user's average rating on all items
- $w_{a,u}$: the similarity between user a and user u

Predicted rating matrix

	0	1	2	3	4	5	6	7	8
0	4.489515	1.424712	3.444746	2.087613	3.512252	3.212174	3.439498	3.192551	3.298408
1	5.274405	2.451446	4.233269	3.117390	4.312363	4.069273	4.184764	4.032432	4.097935
2	3.540935	0.852234	1.987310	1.014415	2.044986	1.751809	2.009870	1.727681	2.078541
3	3.899827	0.471106	3.015265	0.917007	2.872909	2.297834	2.573953	2.248414	2.156151
4	4.075303	0.973355	3.055018	1.587336	3.082620	2.790122	2.989874	2.722689	2.844349
5	3.953429	1.317604	2.933498	1.763504	2.795504	2.616288	2.620987	2.579826	2.515702
6	5.048282	2.256714	3.911012	2.819102	4.047121	3.742852	3.930560	3.777899	3.896936
7	3.988666	1.349240	2.786185	1.576842	2.790481	2.729130	2.555517	2.682731	2.671261
8	3.852521	0.479652	2.970525	1.057247	2.889417	2.453580	2.716724	2.442476	2.507202
9	3.753334	0.741660	2.921326	1.449205	2.597784	2.447671	2.311119	2.392954	2.072932
10	5.454531	2.600169	4.424641	3.282719	4.507595	4.216490	4.384244	4.220429	4.322400

How do we do it?

Cosine Similarity Matrix:

	0	1	2	3	4	5	6	7	8
0	1.000000	0.671274	0.875994	0.723271	0.867604	0.858850	0.858313	0.848783	0.842334
1	0.671274	1.000000	0.572794	0.652670	0.575884	0.598761	0.570187	0.604716	0.582923
2	0.875994	0.572794	1.000000	0.719892	0.903531	0.890338	0.857943	0.886800	0.751093
3	0.723271	0.652670	0.719892	1.000000	0.721027	0.726554	0.677637	0.726151	0.663821
4	0.867604	0.575884	0.903531	0.721027	1.000000	0.897829	0.879509	0.889992	0.789133
5	0.858850	0.598761	0.890338	0.726554	0.897829	1.000000	0.833745	0.904061	0.778952
6	0.858313	0.570187	0.857943	0.677637	0.879509	0.833745	1.000000	0.844189	0.789514
7	0.848783	0.604716	0.886800	0.726151	0.889992	0.904061	0.844189	1.000000	0.768554
8	0.842334	0.582923	0.751093	0.663821	0.789133	0.778952	0.789514	0.768554	1.000000
9	0.845468	0.554882	0.771671	0.643420	0.808614	0.759936	0.799864	0.747167	0.819600
10	0.835470	0.562499	0.723742	0.652981	0.777432	0.759107	0.811620	0.730104	0.801367
11	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Making prediction (Item-based)

$$p_{a,i} = \frac{\sum_{j \in K} r_{a,j} w_{i,j}}{\sum_{j \in K} |w_{i,j}|}$$

Predicted rating matrix

	0	1	2	3	4	5	6	7	8
0	1.419859	2.309058	0.153667	0.086188	0.849639	0.558852	1.957063	0.532820	0.273580
1	1.161997	2.161240	0.148066	0.068088	0.691759	0.589344	1.862274	0.565058	0.207067
2	1.467455	2.401168	0.140114	0.093919	0.884165	0.569674	1.972940	0.530266	0.296208
3	1.301556	2.301175	0.138045	0.071446	0.764794	0.553463	1.885047	0.500943	0.232901
4	1.455499	2.377986	0.141333	0.089311	0.869649	0.544466	1.989984	0.521187	0.285072
5	1.452475	2.403929	0.139973	0.088827	0.873294	0.560987	1.972200	0.549831	0.283874
6	1.464327	2.360843	0.144332	0.085912	0.869628	0.536035	1.965198	0.506532	0.282194
7	1.444466	2.377263	0.137845	0.086622	0.857641	0.553617	1.995098	0.542727	0.280568
8	1.430764	2.321572	0.148831	0.079391	0.850657	0.526886	1.959125	0.514482	0.271213
9	1.464353	2.335122	0.149615	0.087396	0.867139	0.528917	1.949706	0.514296	0.286384
10	1.451773	2.328330	0.147267	0.075485	0.856089	0.516446	1.958239	0.478231	0.261649

Model-based collaborative filtering

Non-negative matrix factorization:

- $\min_{W,H} \|X - WH\|_F$, where $W \geq 0, \ H \geq 0$
- Factorized training data into bases with non-negative values
- W is $p \times r$ matrix and H is $r \times n$ matrix
- p is beer name, r is the base, and n is user name

Predicted rating matrix

	0	1	2	3	4	5	6	7	8
0	0.151387	2.291120	0.378925	0.011792	0.369399	0.903758	2.312437	2.454037	0.000000
1	4.427783	4.435295	0.054312	1.106924	4.734370	1.314400	1.736658	2.438625	3.017126
2	1.913358	4.441350	0.108604	0.042977	1.043610	1.186267	2.151322	1.456367	0.081408
3	4.420813	4.704232	0.098416	1.000775	4.549427	1.293354	2.299141	2.233727	2.720951
4	3.837916	4.503227	0.080942	0.872853	3.953252	1.458598	1.651994	2.675676	2.378569
5	4.744312	3.988868	0.287839	0.961344	4.768576	1.143663	2.924623	1.669561	2.622577
6	3.713260	4.128650	0.445099	0.644222	3.520908	1.448909	1.629379	2.229156	1.786844
7	3.836987	4.459242	0.052443	0.705822	3.510974	1.491469	2.418595	2.544075	1.929604
8	3.666890	5.274035	0.123678	0.534808	3.183742	1.556920	2.628168	2.471622	1.438750
9	3.916236	4.397845	0.328790	0.599716	3.597448	1.345386	2.275433	2.573039	1.650677
10	3.913527	4.692315	0.053493	0.773391	3.745562	1.434489	1.671350	2.740040	2.114327

Model Evaluation

	RMSE	MSE
User-based	2.3532	5.5377
Item-based	2.5068	6.2840
NMF	1.8918	3.5788

Future Work

- Optimize and use package to extract larger datasets
- Find the best base of NMF model using cross validation
- Use more model evaluation techniques
- Make recommendation based on the prediction scores

Conclusion

In our project we demonstrate the full data science process involving data extraction, cleaning, and finally analysis. Although data engineering does not receive as much of a spotlight compared to data analysis, it was very interesting and humbling to see how involved it can be. In the future, we hope both ourselves and others can use our package to extract larger datasets and gain applicable insight for humankind's oldest beverage.

References

BeerAdvocate.com

cambridgespark.com
Implementing your own recommender systems in Python

The End