Technical University of Denmark

Integrating Graph Clustering and Association Rule Mining on the Yelp Dataset for Detection of Global and Local Patterns

02807 Computational Tools for Data Science

Authors

Alejandra Caballero Perez	s231912
Karoline Helene Baarsøe Jørgensen	s204604
Kristine Rigmor Agergaard Andersen	s204608
Petr Boska Nylander	s240466

The Project GitHub Repository

December 5, 2025

Abstract

This project analyzes the Yelp Open Dataset by integrating Louvain community detection, association rule mining, and sentiment analysis to uncover and analyze both global and local patterns. A large user–friendship network was processed and sampled, after which community structures were identified using the Louvain algorithm on the social network of users and their friends. An Apriori algorithm was then applied to restaurant category preferences, both globally and within the six largest found communities, separately for liked and hated reviews. The results show that community association rules are considerably stronger and share little overlap with global rules. The sentiment analysis using a RoBERTa transformer model confirms that the split of reviews closely matches the textual sentiment in each community.

1 Introduction

Yelp is one of many internet platforms that hosts vast amount of user generated content in form of text reviews, images and star based ratings for various businesses and services [1]. According to [2–4], the internet has given a new life to the ancient concept of "word of mouth" letting users share their experiences on a scale that was never seen before. This has a profound influence on user shopping behavior and as byproduct generates large datasets of digital footprints [1–4].

By analyzing these datasets, we can gain an important insight into customer behaviour patterns. For instance, with Market Basket Analysis (MBA), we identify co-occurrences of frequently bought items and association rules (AR), that can be used as pricing and recommendation strategies [5]. Next, we analyze the social dimension of the dataset, by applying social network algorithm to user friends, we look for communities that are more densely connected to each other then the rest of the graph [6]. And lastly, we analyze the degree of positivity and negativity of user reviews.

Our goal in this report is to investigate, to what extent the AR mined from the subset of food and restaurant related businesses differ from those obtained from the communities. In both global and community analysis, we treat each user as a basket and their reviewed restaurant and food categories are the items in the basket. Our assumption is that the global rules might provide a baseline for the general occurring trends, yet they might be too broad to capture the uniqueness of the individual communities. If the following assumption is correct, it could suggest that applying the AR only on the global dataset mean we could miss the important nuances hidden within the communities.

1.1 Yelp dataset

The dataset for this report is originally retrieved from the Yelp Open Dataset [1]. It consists of 5 JSON files with volume of 8.7 GB. For this report we use the files containing approximately 7 million reviews of more than $150,000$ businesses written by approximately $2,000,000$ users.

The files used are *Business*, *Review* and *User*. Single entries from each of the files are shown in Table 2, Table 3 and Table 4 respectively in Appendix A. In the Appendix L, we provide a notebook with a script that automatically downloads the selected files and cleans a small formatting error in the *Review* and includes also all the code used in the report. In addition, we use the Food and Restaurant Categories from Kaggle, to obtain a list of Yelp category titles in preprocessing [7].

1.2 Data processing

In the following two subsections, we outline how the raw Yelp dataset was processed and cleaned to be further used in the analysis in the coming sections.

1.2.1 Graphs

The goal of the Network Analysis is to use the *User* JSON and construct a graph G with social structures, where each node represents a *user_id* and each edge is an *id* of a friend. The

full dataset contains approximately 2 millon users, however half of the users have no friends listed.

Therefore, we extract the greatest connected component (GCC) and filter out the isolated users and small disconnected subgraphs. This process leave us with approximately 900,000 nodes and 7 million edges. In order to visualize and work with the following graph we take a random sample of $30,000$ nodes to create a subgraph H and re-extract the GCC. This results in a final network H with 4132 nodes and 6003 edges shown in Figure 1 we observe that the distribution is highly skewed, where most of the nodes have very few connections.

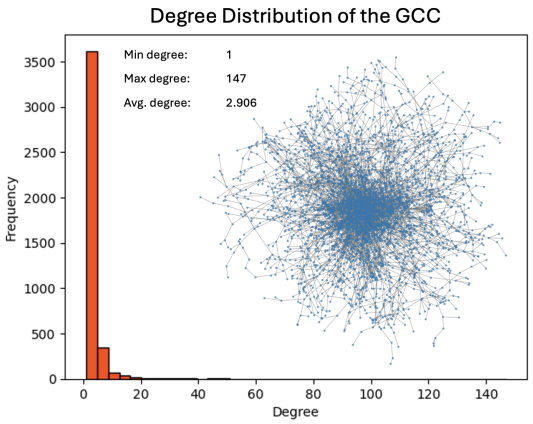


Figure 1. Degree distribution of the GCC, which is highly skewed with many user having few friends. Basic statistics (minimum, maximum, and average degree) are shown in the top left of the plot together with the network visualization on the right.

1.2.2 Apriori

The goal of the data processing for the MBA and the Apriori Algorithm (AA) is to transform the Yelp data into set of transactions, where each transaction consits of user id and set of item categories. We start with *Business* and *Reviews* JSON files and constrain our analysis to Restaurant and Food related categories.

Initially, the *Business* had $37,946$ unique restaurant and food related categories of a following style see Appendix B, Table 5. We reduce the number of categories to 200, by randomly selecting one category per business, see Appendix C Table 6. To justify this simplification, we run 100 Monte Carlo simulations and summarize the distribution in Appendix D Table 7, according to the results, this simplification has a negligible effect on the distribution. The new categories become a new column in the dataset called *simple_category*, with a few dominant categories and a long tail, see Appendix E, Figure 2.

According to [6] the AA is most efficient, when the categories are encoded as integers. Therefore, we create a mapping, from a string category to an unique integer, and inverse mapping, from the integer back to the category.

From the *Business* dataset we extract the columns *business_id* and *simple_category*, and from the *Review* dataset we extract the *user_id, business_id* and *stars*. We inner join these two, to obtain the transaction id *user_id* and set of items of the transaction *simple_category* needed for AA. We split the baskets into two separate sets, *liked* with user reviews stars ≥ 4 and *hated* with < 3 stars, neutral reviews are not considered in the analysis. Lastly, we filter out the baskets with fewer than 2 items. In total, we obtain 3 million *liked* and 1 million *hated* baskets. For an example, see Appendix F, Table 8.

2 Network Construction

To further analyze the structure of the sampled Yelp user graph H obtained in Section 1.2.1, two community-detection methods were applied. Spectral clustering was first used, where

the optimal number of clusters was chosen by maximizing modularity. This resulted in 57 clusters with a modularity of 0.4412, indicating a moderate community structure.

In contrast, the Louvain algorithm (LA) automatically identified 55 communities and achieved a much higher modularity of 0.769, showing that Louvain captures the community structure of this graph significantly better. This is expected as Louvain is designed specifically to maximize modularity and often performs well on large, sparse social networks [8]. We decided to work furthermore with the Louvain Communities as we have focused on the modularity as an indicator for how good the communities are created.

The 6 largest communities can be seen in Figure 3, in Appendix H. From this we can see community 10 is by far the largest, and has the hairball like structure, with a central hub, while the others display more tree like structure, with more small hubs.

3 Market Basket Analysis: Apriori in Global vs Local Communities

For the dataset obtained in Section 1.2.2, we apply the AA with *min support* 0.012 and *confidence* 0.4. This we ensures, that we filter out weak rules and keep rules that are in 1.2% of the total baskets [9]. This leaves us with 1406 *liked* and only 5 *hated* AR. This could be explained by the skewness of the dataset towards number of positive reviews, since we obtained 3 million liked and 1 million hated baskets. Inspecting the rules more closely, we observe that the *liked* rules are more diverse. They include both pairs and triples, while the *hated* rules are more simpler and have lower metrics compared to the *liked*, see Appendix J, Table 11.

As a next part, we apply the AA on the communities, obtained in Section 2 and list the top 3 global association rules for *liked* and *hated* and the same for communities, sorted by the *Lift* measure in the Appendix J, Table 11. To measure the overlap between the global and the community rules, we compute the Jaccard Similarity (JS) between the global baseline and each community. From each dataset we create a set of rules and then calcualte the intersection and union of the two sets, obtaining following similarity, see Table 9. The JS values are very low and for some communities we see a zero overlap, suggesting, that global pattern does not translate to the communities. In addition, we compare the average of the metrics and we find that the mean support, confidence and lift are significantly higher for the communities, than for the global rules, for both liked and hated baskets, suggesting that the obtained community rules are stronger compared to the global baseline, suggesting each community creates pattern, which the global analysis is not able to capture, see Table 1.

Table 1. Metrics for the global and community association rules averaged across the Top 6 communities. The non-averaged metrics for each community are in Appendix K.

Type	**Scope**	**Support**	**Confidence**	**Lift**
Liked	Global	0.025	0.57	2.36
Liked	Communities	0.111	0.78	3.33
Hated	Global	0.013	0.42	1.72
Hated	Communities	0.134	0.86	4.61

4 Sentiment Analysis

Sentiment analysis is a method, which automatically determine emotional tone of a text, where it classifies based on a probability, as positive, negative and neutral. We use RoBERTa based model *cardiffnlp/twitter-roberta-base-sentiment-latest*, which is a pretrained transformer, finetuned for sentiment classification [10]. For each of the reviews within the communities, the model outputs probability score and assigns one of the three labels, with the highest probability [10].

4.1 SA communities

In Appendix I Table 10, we provide a summary of the sentiment analysis for the 6 largest Louvain communities in a confusion matrix like table. It shows row wise the categories, liked (stars ≥ 4), hated (stars < 3) and neutral ($3 \leq$ stars < 4).

Across all six communities, the liked reviews consistently show very high positive sentiment score, mostly above 0.9, indicating that positive reviews contain strongly positive language most of the time. Second, for hated reviews, the negative sentiment is high, but noticeably lower compared to the positive scores in like category, which indicate, that people do not use very negatively charged language, compared to the positive languages. Lastly, the neutral reviews are slightly skewed towards positive sentiment score, than the negative.

To conclude the SA, we observe no large deviation of the sentiment score amongst the communities. This validates the choice of splitting the baskets into the liked and hated as none of them deviates strongly from their respective scores. It also supports the omission of the neutral baskets as their sentiment is slightly skewed towards positive and could possibly create positive like or noisy association rules.

5 Conclusion and Discussion

To conclude, we have evaluated to what extent the association rules mined from the processed Yelp dataset differ from the graph network communities. We have observed, that the global rules capture the overall trends in the data, yet they were not able to capture the uniqueness of each community.

With the LA, we discovered community structures and after applying AA on these 6 communities and comparing them to the global rules, we were able to identify, that the average metrics (support, confidence, lift) were higher. Lastly, with the SA we have confirmed that there were no deviations from the liked and hated categories.

One key limitation to our result is, that the entire analysis is performed only on a single sampled subgraph consisting of 30,000 nodes, drawn from a much larger graph ($1,987,897$ nodes). To be able to fully conclude, we would need to repeat the same analysis n times and aggregate the resulting metrics, or perform the analysis on the entire graph G which will be left for future work.

6 Declaration Of Use Of Generative AI

I/we have used generative AI tools: **yes**

List the generative AI tools you have used and describe how the tools were used:

- ChatGPT - Used to help with debugging and spell check and phrasing in the text, help with plotting, get an idea how to recode the dataset from text to integers and back. For LaTeXcode to make the large tables fast.
- Git Co-Pilot - Used to help with debugging, using the smart suggestions of Co-Pilot, generating docstrings.

At what stage(s) of the process did you use the tool(s)?
When used to help with debugging, it is used to help with code after it has been written. When helped with spell check it is after the text have been written. When helped with phrasing, it was doing the text writing process, if there is something we need help phrasing.

How did you use or incorporate the generated output?
For phrasing, we describe to ChatGPT what we want to communicate, and then it shows a clear proposal of the text that we read and get hints how we can communicate the message we are trying to provide. I have used it for correcting the mistakes in my writing and making my already written text by me, have a better flow of sentences for the reader.

7 Group Contribution

All group members contributed to the project. The responsibilities for each topic is outlined in below table.

Group member	Topic
Petr and Alejandra	Apriori
Kristine and Karoline	Graph Clustering
Kristine and Petr	Sentiment Analysis

For any questions or clarification, please reach out to us via DTU email address given bellow.

Name	**DTU Email**
Karoline Helene Baarsøe Jørgensen	s204604@dtu.dk
Alejandra Caballero Perez	s231912@dtu.dk
Kristine Rigmor Agergaard Andersen	s204608@dtu.dk
Petr Boska Nylander	s240466@dtu.dk

References

[1] Yelp Inc. "Yelp Open Dataset". https://business.yelp.com/data/resources/open-dataset/. Accessed: 1 December 2025. 2025 (page 2).

[2] Nabiha Asghar. "Yelp dataset challenge: Review rating prediction". In: *arXiv preprint arXiv:1605.05362* (2016) (page 2).

[3] Chrysanthos Dellarocas. "The digitization of word of mouth: Promise and challenges of online feedback mechanisms". In: *Management science* 49.10 (2003), pp. 1407–1424 (page 2).

[4] Pei-Yu Chen, Shin-yi Wu, and Jungsun Yoon. "The impact of online recommendations and consumer feedback on sales". In: (2004) (page 2).

[5] Karl Heuer. "Computational Tools for Data Science, Week 5: Frequent Itemsets". Lecture slides, DTU Compute, Technical University of Denmark, 30 September 2025. 2025 (page 2).

[6] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. "Mining of massive data sets". Cambridge university press, 2020 (pages 2, 3).

[7] Yalda Yazdanpanah. "YelpCategoryTitles Dataset". Kaggle dataset. Accessed December 3, 2025. 2025. URL: https://www.kaggle.com/datasets/yaldayazdanpanah/yelpcategorytitles/data (page 2).

[8] Karl Heuer. "Mining social network graphs". PDF file. Accessed: 2025-12-01. 2025. URL: https://courses.compute.dtu.dk/02807/2025/slides/week7.pdf (page 4).

[9] GeeksforGeeks. "Frequent Item set in Data set (Association Rule Mining)". https://www.geeksforgeeks.org/data-science/frequent-item-set-in-data-set-association-rule-mining/. Last updated 03 Feb 2023, accessed on 01 Dec 2025. 2023 (page 4).

[10] GeeksforGeeks contributors. "Sentiment Analysis using HuggingFace's RoBERTa Model". https://www.geeksforgeeks.org/nlp/sentiment-analysis-using-huggingfaces-roberta-model/. Last updated 23 Jul 2025, accessed. 2025 (page 5).

[11] Meva Nokta. "Association Rule Based Recommender System". https://www.kaggle.com/code/noktameva/association-rule-based-recommender-system. Kaggle notebook, accessed 4 December 2025. 2023 (page 13).

A Examples from Yelp dataset

A.1 User JSON

Table 2. Example of the data attributes from a single user from the Yelp dataset.

Attribute	Value
user_id	qVc8ODYU5SZjKXVBgXdI7w
name	Walker
review_count	585
yelping_since	2007-01-25 16:47:26
useful	7217
funny	1259
cool	5994
elite	2007
friends	NSCy54eWehBJyZdG2iE84w, ...
fans	267
average_stars	3.910000
compliment_hot	250
compliment_more	65
compliment_profile	55
compliment_cute	56
compliment_list	18
compliment_note	232
compliment_plain	844
compliment_cool	467
compliment_funny	467
compliment_writer	239
compliment_photos	180

A.2 Review JSON

Table 3. Example of the data attributes from a single review from the Yelp dataset.

Attribute	Value
review_id	KU_O5udG6zpxOg-VcAEodg
user_id	mh_-eMZ6K5RLWhZyISBhwA
business_id	XQfwVwDr-v0ZS3_CbbE5Xw
stars	3.000000
useful	0
funny	0
cool	0
text	If you decide to eat here, just be aware it is going to take about 2 hours ...
date	2018-07-07 22:09:11

A.3 Business JSON

Table 4. Example of he data attributes from a single business from the Yelp dataset.

Attribute	**Value**
business_id	MTSW4McQd7CbVtyjqoe9mw
name	St Honore Pastries
address	935 Race St
city	Philadelphia
state	PA
postal_code	19107
latitude	39.955505
longitude	-75.155564
stars	4.000000
review_count	80
is_open	1
attributes	{'RestaurantsDelivery': 'False', 'OutdoorSeating': 'False', ... ,}
categories	Restaurants, Food, Bubble Tea, Coffee & Tea, Bakeries
hours	{'Monday': '7:0-20:0', ...}

B Business Categories

Table 5. Ten examples of the column Categories from the Yelp dataset.

Index	Categories
0	Restaurants, Food, Bubble Tea, Coffee & Tea, Bakeries
1	Brewpubs, Breweries, Food
2	Burgers, Fast Food, Sandwiches, Food, Ice Cream & Frozen Yogurt, Restaurants
3	Pubs, Restaurants, Italian, Bars, American (Traditional), Nightlife, Greek
4	Ice Cream & Frozen Yogurt, Fast Food, Burgers, Restaurants, Food
5	Vietnamese, Food, Restaurants, Food Trucks
6	American (Traditional), Restaurants, Diners, Breakfast & Brunch
7	Food, Delis, Italian, Bakeries, Restaurants
8	Sushi Bars, Restaurants, Japanese
9	Korean, Restaurants
10	Coffee & Tea, Food, Cafes, Bars, Wine Bars, Restaurants, Nightlife

C Simple Categories

Table 6. Same ten examples as in the previous Table 5, but simplified by our method of selecting one random food category, to reduce the size of the categories.

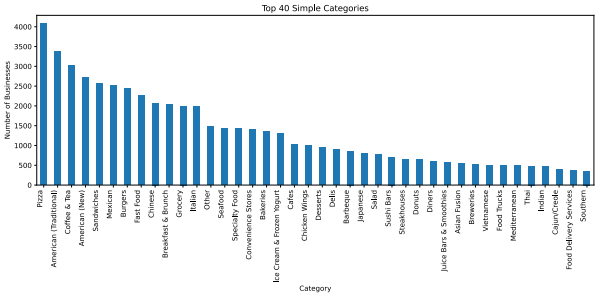
Index	Simple category
0	Coffee & Tea
1	Brewpubs
2	Ice Cream & Frozen Yogurt
3	American (Traditional)
4	Ice Cream & Frozen Yogurt
5	Food Trucks
6	Breakfast & Brunch
7	Italian
8	Japanese
9	Korean
10	Cafes

D Categories Distribution

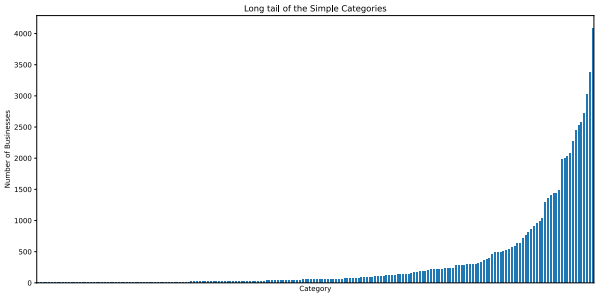
Table 7. Result of the stability analysis of the distribution of simplified category share. For each category c we compute mean share μ_c and standard deviation σ_c for 100 runs with diffeerent seed. The table summarizes statistics over the categories c, the mean, standard deviation and range of μ_c adn σ_c. We conclude, tha the distribution is very stable.

Statistic over category columns c	μ_c	σ_c
Mean	4.98×10^{-3}	8.62×10^{-19}
Std. deviation	1.03×10^{-2}	2.41×10^{-18}
Range $[\min, \max]$	$[1.55 \times 10^{-5},\ 5.87 \times 10^{-2}]$	$[0,\ 2.09 \times 10^{-17}]$

E Long tail and Frequency of Top highest 40 Categories



(a) Frequency of top 40 simple categories



(b) Long tail of the simple categories

Figure 2. Frequency of top 40 simple categories (left) and long tail of the simple categories (right).

F Encoded and Decoded Examples of the Dataset for Apriori Algorithm

Table 8. Example of the text and integer mapping for the Apriori Algorithm.

user_id	**Encoded items (IDs)**	**Decoded items (categories)**
–2PmXbF47D87OstH1jqA	[3, 4, 46, 74, 121, 151, 155, 168]	[American (New), American (Traditional), Comfort Food, Food Trucks, Mexican, Sandwiches, Seafood, Southern]
–UgP949okyCDuB5zUsSA	[3, 29, 74, 104, 107]	[American (New), Cafes, Food Trucks, Italian, Juice Bars & Smoothies]
–r61b7EpPKbb4Uzm5tA	[7, 10]	[French, Italian]
–2bpEsv5yr-2hAP7sZZAIA	[3, 4, 104]	[American (New), American (Traditional), Italian]
–4AjktZiHowElBCMd4CZA	[13, 11, 25, 104, 121, 151, 185, 199]	[American (New), Bakeries, Burgers, Italian, Mexican, Sandwiches, Thai, Wineries]

G Jaccard Similarity of the Communities and Global Association Rules

Table 9. Jaccard similarity between global and 6 communities Aprioru rules for liked and hated baskets.

Community index	Jaccard Similarity Liked	Jaccard Similarity Hated
10	0.1197	0.0027
5	0.0281	0.0000
0	0.0208	0.0714
4	0.0109	0.0000
3	0.0402	0.0021
2	0.0000	0.0000

H Visualization of the Top 6 Louvian communities.

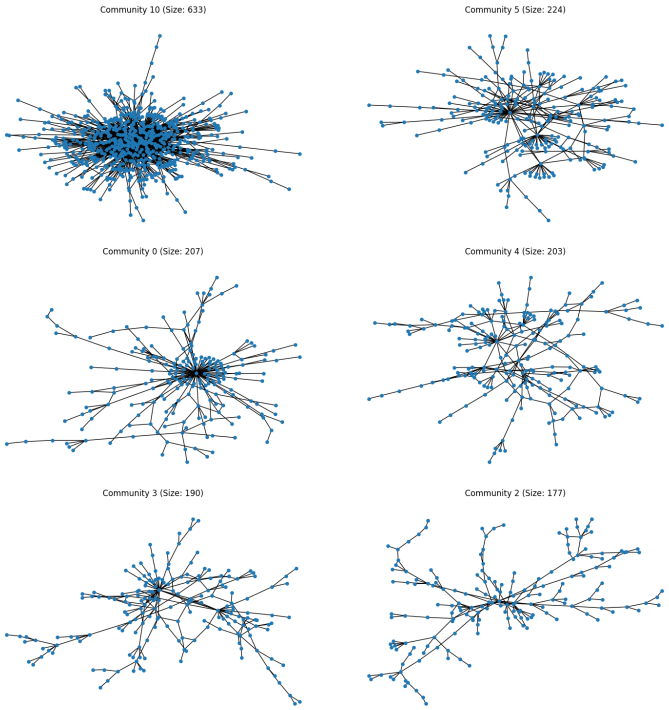


Figure 3. A visualization of the six largest Louvain communities in the sampled Yelp user graph. Each subplot shows the internal structure of a community, where tightly connected cores are surrounded by loosely connected peripheral users.

I Sentiment Confusion Matrices

Table 10. Sentiment scores (negative, neutral, positive) for Yelp labels across the six largest communities.

Community	Yelp label	Negative	Neutral	Positive
Top 1	hated	0.664223	0.149560	0.186217
	liked	0.027936	0.063373	0.908691
	neutral	0.171875	0.173713	0.654412
Top 2	hated	0.734483	0.124138	0.141379
	liked	0.020214	0.055291	0.924495
	neutral	0.244318	0.190341	0.565341
Top 3	hated	0.737013	0.149351	0.113636
	liked	0.033514	0.056757	0.909730
	neutral	0.270073	0.145985	0.583942
Top 4	hated	0.756757	0.156156	0.087087
	liked	0.023856	0.058672	0.917473
	neutral	0.300395	0.130435	0.569170
Top 5	hated	0.681319	0.131868	0.186813
	liked	0.025086	0.076397	0.898518
	neutral	0.195000	0.215000	0.590000
Top 6	hated	0.827586	0.086207	0.086207
	liked	0.009288	0.015480	0.975232
	neutral	0.360000	0.160000	0.480000

J Association Rules

The association rules are mathematically written as conditional statements, i.e. $A \implies B$ and in the following Table 11, A is named **Antecedent** and B is **Consequent** [11].

K Jaccard Similarities

Table 12. Support, confidence, and lift for liked vs hated rules for the global and Top 6 communities.

	Liked rules			Hated rules		
Scope	**Support**	**Confidence**	**Lift**	**Support**	**Confidence**	**Lift**
Global	0.025	0.574	2.362	0.013	0.422	1.724
Top 1 Com.	0.072	0.764	3.003	0.073	0.839	7.262
Top 2 Com.	0.113	0.812	3.415	0.121	0.867	2.877
Top 3 Com.	0.111	0.793	3.363	0.129	0.808	2.896
Top 4 Com.	0.212	0.810	2.356	0.222	0.732	2.052
Top 5 Com.	0.076	0.793	4.170	0.114	0.908	5.877
Top 6 Com.	0.081	0.738	3.661	0.143	1.000	6.712

L Jupyter Notebook

FINAL_NOTEBOOK

December 5, 2025

1 Setup the environemnt and install all packages

1.1 Setup

1. Clone the repository

```
git clone https://github.com/Cerlog/02807-project
```

2. Create conda environment

```
conda create -n 02807-project python=3.11.14 -y
```

3. Activate the environment

```
conda activate 02807-project
```

4. Change the directories

```
cd 02807-project
```

5. Install the requirements

```
pip install -r requirements.txt
```

---

2 Integrating Graph Clustering and Association Rule Mining on the Yelp Dataset for Detection of Global and Local Patterns

[85]:
```
# imports
import os
import json
import nltk
import random
import community.community_louvain as community_louvain
from networkx.algorithms.community import modularity
import sys
import importlib
import pickle
import torch
```

```
import tqdm as notebook_tqdm
import pandas as pd
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
import pyarrow.parquet as pq

from pathlib import Path
from nltk.sentiment import SentimentIntensityAnalyzer
from collections import Counter
from collections import defaultdict
from src.functions import *
from transformers import AutoTokenizer
from transformers import pipeline
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
import matplotlib.cm as cm
#
## set paths
PROJECT_ROOT = Path().resolve()
SRC_DIR = PROJECT_ROOT / "src"
sys.path.append(str(SRC_DIR))
sys.path.insert(0, str(SRC_DIR))

# functions
import data as d
import apriori as ap
import preprocessing as pr

# set random seed for reproducibility
SEED = 1337
os.environ["PYTHONHASHSEED"] = str(SEED)
random.seed(SEED)
np.random.seed(SEED)
rng = np.random.default_rng(SEED)


# Define base directory
DATA_DIR_RAW = Path("data/raw")
DATA_DIR_PROC = Path("data/processed")

# Define file paths
USERS   = DATA_DIR_RAW / "yelp_academic_dataset_user.json"
REVIEWS = DATA_DIR_RAW / "yelp_academic_dataset_review.json"
REVIEWS_CLEAN = DATA_DIR_RAW / "review_clean.ndjson"
BIZ     = DATA_DIR_RAW / "yelp_academic_dataset_business.json"
```

```
# categories of restaurants
FOOD = DATA_DIR_RAW / "Food.txt"
RESTAURANTS = DATA_DIR_RAW / "restaurants.txt"


# pull the datasets from google drive
DATASET_URLS = [
    "https://drive.google.com/file/d/163KgvKNYPTV5_fArvlBiNkqb6rCDkk-Z/view?
 ↪usp=drive_link",
    "https://drive.google.com/file/d/1O5kEcRxvnO1da74y9y5AE92h_QlOfjmL/view?
 ↪usp=drive_link",
    "https://drive.google.com/file/d/1ox-qrD1sSKalbu25FIbu-kYGMvkn3Y3k/view?
 ↪usp=drive_link",
    "https://drive.google.com/file/d/1cJx0UUVxwsKL8DHpwrGoqBSlRA2v4I2f/view?
 ↪usp=drive_link",
    "https://drive.google.com/file/d/10wpi7RzZTpt93_7uRZcshvA5X3ek3_Kc/view?
 ↪usp=drive_link"
]

# https://www.kaggle.com/datasets/yaldayazdanpanah/yelpcategorytitles/data
#https://business.yelp.com/data/resources/open-dataset/
```

3 Get the dataset

[2]:
```
d.download_dataset(DATASET_URLS, DATA_DIR_RAW)
```

```
2025-12-05 09:57:33.759 | INFO     |
data:download_dataset:35 - Downloading from https
://drive.google.com/file/d/163KgvKNYPTV5_fArvlBiNkqb6rCDkk-
Z/view?usp=drive_link…
Downloading…
From (original):
https://drive.google.com/uc?id=163KgvKNYPTV5_fArvlBiNkqb6rCDkk-Z
From (redirected): https://drive.google.com/uc?id=163KgvKNYPTV5_fArvlBiNkqb6rCDk
k-Z&confirm=t&uuid=6274ec39-55cd-4509-bb16-83ce02c3921e
To: /home/pnylander/DTU/4thSemester/02807/02807-
project/yelp_academic_dataset_user.json
100%|      | 3.36G/3.36G [01:55<00:00, 29.2MB/s]
2025-12-05 09:59:32.870 | SUCCESS  |
data:download_dataset:42 -  Downloaded to
'data/raw/yelp_academic_dataset_user.json'
2025-12-05 09:59:32.871 | WARNING  |
data:clean_review_json:10 - Can't find
'review.json' to clean.
```

```
2025-12-05 09:59:32.872 | INFO     |
data:download_dataset:35 - Downloading from https
://drive.google.com/file/d/1O5kEcRxvnO1da74y9y5AE92h_QlOfjmL/view?usp=drive_link
…
Downloading…
From (original):
https://drive.google.com/uc?id=1O5kEcRxvnO1da74y9y5AE92h_QlOfjmL
From (redirected): https://drive.google.com/uc?id=1O5kEcRxvnO1da74y9y5AE92h_QlOf
jmL&confirm=t&uuid=365528c8-eb4d-4f20-a46f-5b1e6b524388
To: /home/pnylander/DTU/4thSemester/02807/02807-
project/yelp_academic_dataset_review.json
100%|      | 5.34G/5.34G [02:58<00:00, 30.0MB/s]
2025-12-05 10:02:34.403 | SUCCESS  |
data:download_dataset:42 -  Downloaded to
'data/raw/yelp_academic_dataset_review.json'
2025-12-05 10:02:34.404 | WARNING  |
data:clean_review_json:10 - Can't find
'review.json' to clean.
2025-12-05 10:02:34.405 | INFO     |
data:download_dataset:35 - Downloading from
https://drive.google.com/file/d/1ox-qrD1sSKalbu25FIbu-
kYGMvkn3Y3k/view?usp=drive_link…
Downloading…
From (original): https://drive.google.com/uc?id=1ox-qrD1sSKalbu25FIbu-
kYGMvkn3Y3k
From (redirected): https://drive.google.com/uc?id=1ox-qrD1sSKalbu25FIbu-
kYGMvkn3Y3k&confirm=t&uuid=fc62d3da-e99d-430d-bb04-4560c43e28a1
To: /home/pnylander/DTU/4thSemester/02807/02807-
project/yelp_academic_dataset_business.json
100%|      | 119M/119M [00:03<00:00, 31.0MB/s]
2025-12-05 10:02:41.211 | SUCCESS  |
data:download_dataset:42 -  Downloaded to
'data/raw/yelp_academic_dataset_business.json'
2025-12-05 10:02:41.212 | WARNING  |
data:clean_review_json:10 - Can't find
'review.json' to clean.
2025-12-05 10:02:41.212 | INFO     |
data:download_dataset:35 - Downloading from https
://drive.google.com/file/d/1cJx0UUVxwsKL8DHpwrGoqBSlRA2v4I2f/view?usp=drive_link
…
Downloading…
From: https://drive.google.com/uc?id=1cJx0UUVxwsKL8DHpwrGoqBSlRA2v4I2f
```

```
To: /home/pnylander/DTU/4thSemester/02807/02807-project/Food.txt
100%|      | 791/791 [00:00<00:00, 3.14MB/s]
2025-12-05 10:02:43.362 | SUCCESS  |
data:download_dataset:42 -  Downloaded to

'data/raw/Food.txt'
2025-12-05 10:02:43.363 | WARNING  |
data:clean_review_json:10 - Can't find

'review.json' to clean.
2025-12-05 10:02:43.364 | INFO     |
data:download_dataset:35 - Downloading from https

://drive.google.com/file/d/10wpi7RzZTpt93_7uRZcshvA5X3ek3_Kc/view?usp=drive_link

…
Downloading…
From: https://drive.google.com/uc?id=10wpi7RzZTpt93_7uRZcshvA5X3ek3_Kc
To: /home/pnylander/DTU/4thSemester/02807/02807-project/restaurants.txt
100%|      | 1.59k/1.59k [00:00<00:00, 2.63MB/s]
2025-12-05 10:02:45.730 | SUCCESS  |
data:download_dataset:42 -  Downloaded to

'data/raw/restaurants.txt'
2025-12-05 10:02:45.732 | WARNING  |
data:clean_review_json:10 - Can't find

'review.json' to clean.
2025-12-05 10:02:45.733 | INFO     |
data:download_dataset:50 - Downloading dataset

from Google Drive…
Downloading…
From: https://drive.google.com/uc?id=10wpi7RzZTpt93_7uRZcshvA5X3ek3_Kc
To: /home/pnylander/DTU/4thSemester/02807/02807-project/restaurants.txt
100%|      | 1.59k/1.59k [00:00<00:00, 5.63MB/s]
2025-12-05 10:02:47.763 | SUCCESS  |
data:download_dataset:53 -  Downloaded JSON

to 'restaurants.txt'
```

4 Processing the dataset

Here we load the `user`, `review`, and `business` dataset from the Yelp JSON. Then we drop NaN values.

[86]:
```python
# import the data
user = pd.read_json(USERS, lines=True, dtype={"user_id": str}, engine="pyarrow")
review = pd.read_json(REVIEWS_CLEAN, lines=True, engine="pyarrow")
business = pd.read_json(BIZ, lines=True, dtype_backend="pyarrow")
```

```
# drop all the nans within categories
business = business.dropna(subset=["categories"])
business = business.reset_index(drop=True) # reset the index after dropping␣
 ↪nans
```

We are intersted only in the reviews of businesses which are food and restaurants related, so therefore we filter the business dataset to only contain those entries.

[ ]:
```
# keep only food and restaurant businesses
is_food_rest = business["categories"].str.contains(r"\b(?:Restaurants|Food)\b",␣
 ↪case=False, na=False)
# reset index after filtering
business = business.loc[is_food_rest].copy().reset_index(drop=True)
```

[5]:
```
# save the business for later use
#business.to_json(DATA_DIR_PROC / "business_food_restaurants.json",␣
 ↪orient="records", lines=True)
```

Now we have only the food and restaurant related businesses in the business dataframe. The problem is however, that many of the reviews are for businesses contain multiple categories (e.g. *Coffee & Tea, Bubble Tea, Bakeries*), therefore we need to reduce the categories to only one per business. From the the kaggle dataset (https://www.kaggle.com/datasets/yaldayazdanpanah/yelpcategorytitles/data), we have all the categories within **Restaurants** and **Foods** that are included in the yelp dataset, from which we will build a dictionary of the categories in the entire dataset in order to reduce the multiple categories into only one overall category per business.

[6]:
```
# creating the categories in order list
pr.CATEGORIES_IN_ORDER = pr.get_categories(RESTAURANTS, FOOD)
```

Creating new column based on the reduced categories.

[7]:
```
random.seed(SEED)
np.random.seed(SEED)
business["simple_category"] = business["categories"].apply(pr.simplify_random)
```

[ ]:
```
# save the dataframe for later use
#business.to_json(DATA_DIR_PROC / "business_food_restaurants_simple_categories.
 ↪json", orient="records", lines=True)
```

In order to see how this affects the distribution of the dataset, i.e. of choosing Bubble Tea instead of Coffee & Tea, we run a a Monte Carlo simulation to see see if the categories are affected by this simplification.

[8]:
```
pr.test_distribution(business, n_runs=100, run_index=SEED)
```

```
mean:
  Mean: 4.98e-03
  Std Dev: 1.03e-02
  Range: [1.55e-05, 5.87e-02]
```

```
std:
  Mean: 8.62e-19
  Std Dev: 2.41e-18
  Range: [0.00e+00, 2.09e-17]
```

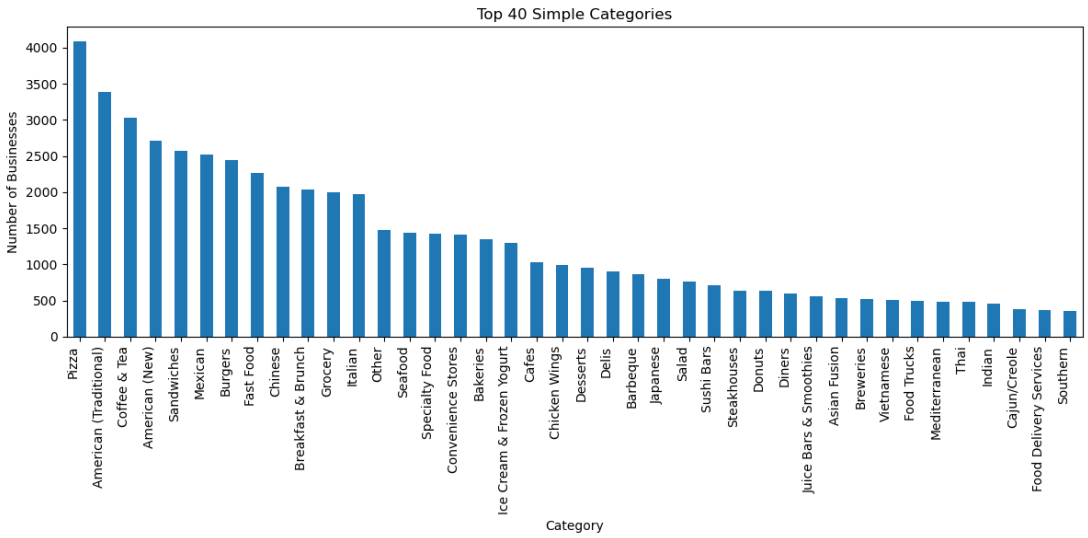
From the 100 trials (plan was to do more, but it took much longer time than expected), we can see that when choosen at random, the distribution odes not change significantly, therefore we can justify the simplification of the categories.

[9]:
```
print(f"Number of unique simplified categories:␣
  ↪{len(business['simple_category'].unique())}")
```

```
Number of unique simplified categories: 200
```

4.0.1 Plot of the top 40 categories

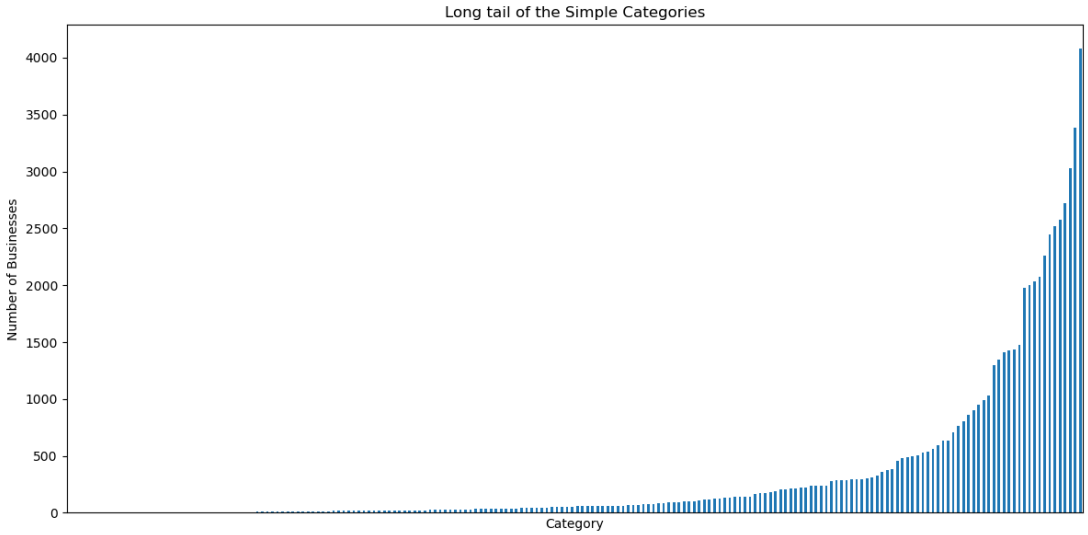
[10]:
```
top_k_simple_categories = 40
category_counts = business["simple_category"].value_counts().
  ↪sort_values(ascending=False)
plt.figure(figsize=(12,6))
category_counts.head(top_k_simple_categories).plot(kind='bar')
plt.title(f"Top {top_k_simple_categories} Simple Categories")
plt.xlabel("Category")
plt.ylabel("Number of Businesses")
plt.xticks(rotation=90, ha="right")
plt.tight_layout()
plt.savefig("figures/simple_categories_distribution.svg",dpi=300)
plt.show()
```



4.0.2 Plot of all the simple categories

We notice very long tail distribution.

[11]:
```
bottom_k_simple_categories = None
category_counts = business["simple_category"].value_counts().
 ↪sort_values(ascending=True)
plt.figure(figsize=(12,6))
ax = category_counts.head(bottom_k_simple_categories).plot(kind='bar')
ax.tick_params(axis='x', which='both', bottom=False, labelbottom=False)
plt.title(f"Long tail of the Simple Categories")
plt.xlabel("Category")
plt.ylabel("Number of Businesses")
plt.tight_layout()
plt.savefig("figures/simple_categories_least_common.svg", dpi=300)
plt.show()
```



Now, that we have the reduced categories, we need to process the data into a form, that can be used by the apriori algorithm.

[12]:
```
# Load and prepare business mapping
#business.to_parquet(DATA_DIR_PROC / "business_processed.parquet", index=False)
```

[87]:
```
biz_tab = pq.read_table(DATA_DIR_PROC / "business_processed.parquet",␣
 ↪columns=["business_id", "simple_category"])
biz_proc = biz_tab.to_pandas()
```

The Apriori algorithm is most efficient, if we assign every category an unique integer ID and threffore we need to create a mapping from category to ID and vice versa.

[88]:
```
# create a list of alphabetically sorted unique categories
cats = sorted(biz_proc["simple_category"].drop_duplicates().to_list())
#print(f"Unique simplified categories: {cats}")
# create dictionary mapping from category to ID
cat_to_id = {cat: idx for idx, cat in enumerate(cats)}
#print(f"Category to ID mapping: {cat_to_id}")
# create reverse mapping from ID to category
id2cat_map = {idx: cat for cat, idx in cat_to_id.items()}
#print(f"ID to Category mapping: {id2cat_map}")

# Save category vocabulary
pd.DataFrame({"cat_id": range(len(cats)), "simple_category": cats}).to_parquet(
    DATA_DIR_PROC / "cat_vocab.parquet", index=False
)
```

Now, we create a lookup table for the businesses to their assigned category IDs.

[89]:
```
# Create business mapping
biz_map = (biz_proc.assign(cat_id=lambda d: d["simple_category"].
 ↪map(cat_to_id))[["business_id", "cat_id"]].drop_duplicates("business_id").
 ↪set_index("business_id"))
biz_map.head()
print(f"Created mappings for {len(cats)} categories")
biz_map
```

```
Created mappings for 201 categories
```

[89]:

business_id	cat_id
MTSW4McQd7CbVtyjqoe9mw	43
mWMc6_wTdE0EUBKIGXDVfA	21
CF33F8-E6oudUQ46HnavjQ	67
k0hlBqXX-Bt0vf1op7Jr1w	82
bBDDEgkFA1Otx9Lfe7BZUQ	67
…	…
cM6V90ExQD6KMSU3rRB5ZA	29
1jx1sfgjgVg0nM6n3p0xWA	44
WnT9NIzQgLlILjPT0kEcsQ	121
2O2K6SXPWv56amqxCECd4w	46
hn9Toz3s-Ei3uZPt7esExA	122

[64616 rows x 1 columns]

Now we create a sets of liked and hated category IDs per user. Where we for each user build a set of categories they liked and disliked based on the star ratings in their reviews.

[ ]:
```
liked_ids = defaultdict(set)
hated_ids = defaultdict(set)
```

```
usecols = ["user_id", "business_id", "stars"]
```

Because of the size of the review dataset, we process the reviews in chunks of 500.000 entries at a time to avoid memory issues.

In this code, we crate two baskets for each user, one with the categories they liked (gave review stars >= 4) and one with the categories they hated (gave review stars < 3).

[ ]:
```
for chunk in pd.read_json(REVIEWS_CLEAN, lines=True, dtype_backend="pyarrow",␣
↪chunksize=500_000):
    r = (chunk[usecols]
         .dropna(subset=["user_id", "business_id", "stars"])
         .copy())

    # Convert to string
    r["user_id"] = r["user_id"].astype("string[pyarrow]")
    r["business_id"] = r["business_id"].astype("string[pyarrow]")


    # Split by stars
    high = r[r["stars"] >= 4.0]
    low = r[r["stars"] < 3.0]  #

    # Join with business mapping and accumulate
    if len(high):
        high = high.join(biz_map, on="business_id", how="inner")
        for uid, cats in high.groupby("user_id")["cat_id"]:
            liked_ids[str(uid)].update(cats.unique())

    if len(low):
        low = low.join(biz_map, on="business_id", how="inner")
        for uid, cats in low.groupby("user_id")["cat_id"]:
            hated_ids[str(uid)].update(cats.unique())
```

Now, we created the dataframes and keep the baskets that have at least 2 items.

[ ]:
```
# Create basket dataframes
liked_baskets_ids = pd.DataFrame({
    "user_id": list(liked_ids.keys()),
    "items": [sorted(list(s)) for s in liked_ids.values()]
})

hated_baskets_ids = pd.DataFrame({
    "user_id": list(hated_ids.keys()),
    "items": [sorted(list(s)) for s in hated_ids.values()]
})

#
```

```
liked_baskets_ids = liked_baskets_ids[liked_baskets_ids["items"].map(len) >= 2]
hated_baskets_ids = hated_baskets_ids[hated_baskets_ids["items"].map(len) >= 2]

# Save all basket variants
liked_baskets_ids.to_parquet(DATA_DIR_PROC / "baskets_liked_ids.parquet",␣
 ↪index=False)
hated_baskets_ids.to_parquet(DATA_DIR_PROC / "baskets_hated_ids.parquet",␣
 ↪index=False)

print(f"Liked baskets: {len(liked_baskets_ids)} users (with 2+ items)")
print(f"Hated baskets: {len(hated_baskets_ids)} users (with 2+ items)")
```

```
Liked baskets: 326145 users (with 2+ items)
Hated baskets: 101028 users (with 2+ items)
```

[ ]:
```
liked_baskets_ids.head()
```

[ ]:

	user_id	items
0	---2PmXbF47D870stH1jqA	[3, 4, 46, 74, 121, 151, 155, 168]
1	---UgP94gokyCDuB5zUssA	[3, 29, 74, 104, 107]
2	---r61b7EpVPkb4UVme5tA	[75, 104]
3	--2bpE5vyR-2hAP7sZZ4lA	[3, 4, 104]
4	--4AjktZiHowEIBCMd4CZA	[3, 11, 25, 104, 121, 151, 185, 199]

5 Apriori on the entire dataset

[ ]:
```
CAT_VOCAB     = DATA_DIR_PROC / "cat_vocab.parquet"
BASKETS_LIKED = DATA_DIR_PROC / "baskets_liked_ids.parquet"
BASKETS_HATED = DATA_DIR_PROC / "baskets_hated_ids.parquet"
OUT_DIR       = DATA_DIR_PROC / "apriori_results"
```

[ ]:
```
# Load category mapping
id2cat_df  = pd.read_parquet(CAT_VOCAB)
id2cat_map = id2cat_df.set_index("cat_id")["simple_category"].to_dict()
```

[ ]:
```
MIN_SUPPORT = 0.012
MIN_CONF    = 0.4

df_rules_liked, transactions_l = ap.run_apriori("liked", BASKETS_LIKED,␣
 ↪MIN_SUPPORT, MIN_CONF, OUT_DIR, id2cat_map)
df_rules_hated, transactions_h = ap.run_apriori("hated", BASKETS_HATED,␣
 ↪MIN_SUPPORT, MIN_CONF, OUT_DIR, id2cat_map)

print("\n" + "="*50)
print(f"Rules liked: {len(df_rules_liked)}")
print(f"Rules hated: {len(df_rules_hated)}")
print("="*50)
```

```
Running Apriori for liked
Generated 1406 liked rules
Saved 1406 liked rules
Running Apriori for hated
Generated 5 hated rules
Saved 5 hated rules

================================================
Rules liked: 1406
Rules hated: 5
================================================
```

We have obtained 1406 association rules for the liked categories and 5 for the hated categories using the apriori algorithm.

Now, we filter out the liked rules to narrow down the numbers.

[90]:
```
rules_liked_global = ap.filter_rules(df_rules_liked, min_support=0.02,␣
↪min_confidence=0.5, min_lift=2, max_antecedent=3, color="Greens",␣
↪save_path="figures/rules_liked_global_filtered.html")
```

[91]:
```
rules_liked_global
```

[91]:

	Antecedent	Consequent	Support	Confidence	Lift	\
949	(3, 4, 19)	(121,)	0.020687	0.577456	3.132330	
950	(3, 4, 121)	(19,)	0.020687	0.616108	3.121892	
951	(3, 19, 121)	(4,)	0.020687	0.704795	2.889316	
952	(4, 19, 121)	(3,)	0.020687	0.735929	2.886482	
539	(138, 151)	(121,)	0.021579	0.504516	2.736677	
..	…	…	…	…	…	
544	(121, 138)	(3,)	0.026709	0.550319	2.158477	
514	(4, 19)	(3,)	0.035825	0.548107	2.149801	
83	(4, 121)	(3,)	0.033577	0.547331	2.146757	
349	(4, 155)	(3,)	0.028693	0.543375	2.131240	
513	(3, 19)	(4,)	0.035825	0.516123	2.115855	

	Antecedent_decoded	\
949	(American (New), American (Traditional), Break…	
950	(American (New), American (Traditional), Mexican)	
951	(American (New), Breakfast & Brunch, Mexican)	
952	(American (Traditional), Breakfast & Brunch, M…	
539	(Pizza, Sandwiches)	
..	…	
544	(Mexican, Pizza)	
514	(American (Traditional), Breakfast & Brunch)	
83	(American (Traditional), Mexican)	
349	(American (Traditional), Seafood)	
513	(American (New), Breakfast & Brunch)	

```
            Consequent_decoded
949                    (Mexican,)
950         (Breakfast & Brunch,)
951  (American (Traditional),)
952           (American (New),)
539                    (Mexican,)
..                           …
544           (American (New),)
514           (American (New),)
83            (American (New),)
349           (American (New),)
513  (American (Traditional),)

[69 rows x 7 columns]
```

Filter the rules based on the highest lift values.

[92]:
```
rules_liked_lift = (
    rules_liked_global
    .sort_values(["Lift"], ascending=False)
    .head(5)
)

num_cols = ["Support", "Confidence", "Lift"]


rules_liked_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Greens, subset=num_cols)
```

[92]: <pandas.io.formats.style.Styler at 0x7f3c7e5bef90>

Do the same for hated, but as we got only 5 rules, we keep them all.

[94]:
```
rules_hated_global = ap.filter_rules(df_rules_hated, min_support=0.012,␣
↪min_confidence=0.4, min_lift=1.6, max_antecedent=3, color="Reds",␣
↪save_path="figures/rules_hated_global_filtered.html")
```

[95]:
```
rules_hated_global.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Reds, subset=num_cols)
```

[95]: <pandas.io.formats.style.Styler at 0x7f3c7e406d90>

6 Grap Network

[145]:
```
with open(USERS, 'r') as f:
    user_counts = Counter(json.loads(line)['user_id'] for line in f)
```

[146]:
```
# number of unique users
num_unique_users = len(user_counts)
print(f'Number of unique users: {num_unique_users}')
```

```
Number of unique users: 1987897
```

[147]:
```
# create graph
G = nx.Graph()
G.add_nodes_from(user_counts.keys())
with open(USERS, 'r') as f:
    for line in f:
        user = json.loads(line)
        user_id = user['user_id']
        friends = user['friends'].split(', ') if user['friends'] != 'None' else␣
↪[]
        for friend in friends:
            if friend in user_counts:
                G.add_edge(user_id, friend)
```

[150]:
```
degrees = [G.degree(node) for node in G.nodes()]
sorted(degrees)
no_friends = sum(count == 0 for count in degrees)
print(f'Number of users with no friends in the network: {no_friends}')
```

```
Number of users with no friends in the network: 1081718
```

[151]:
```
number_of_nodes_full_network = G.number_of_nodes()
number_of_edges_full_network = G.number_of_edges()
print(f'Number of nodes in the graph: {number_of_nodes_full_network}')
print(f'Number of edges in the graph: {number_of_edges_full_network}')
```

```
Number of nodes in the graph: 1987897
Number of edges in the graph: 7305874
```

Because we have so many single nodes, we'd rather look at the greatest component in the network, thus filtering out the nodes not connected to any and smaller subgraphs.

[152]:
```
# taking the greatest component
Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
G = G.subgraph(Gcc[0])
```

[153]:
```
number_of_nodes_greatest_component = G.number_of_nodes()
number_of_edges_greatest_component = G.number_of_edges()
print(f'Number of nodes in the graph: {number_of_nodes_greatest_component}')
```

```
print(f'Number of edges in the graph: {number_of_edges_greatest_component}')
```

```
Number of nodes in the graph: 892152
Number of edges in the graph: 7298492
```

[154]:
```
removed_nodes = number_of_nodes_full_network-number_of_nodes_greatest_component
print(f'Number of nodes filtered away is␣
↪{number_of_nodes_full_network-number_of_nodes_greatest_component}, where␣
↪{removed_nodes-no_friends} had at least one friend, but were not connected␣
↪to the greatest component')
```

```
Number of nodes filtered away is 1095745, where 14027 had at least one friend,
but were not connected to the greatest component
```

[155]:
```
removed_edges = number_of_edges_full_network-number_of_edges_greatest_component
print(f'Number of edges filtered away is {removed_edges}')
```

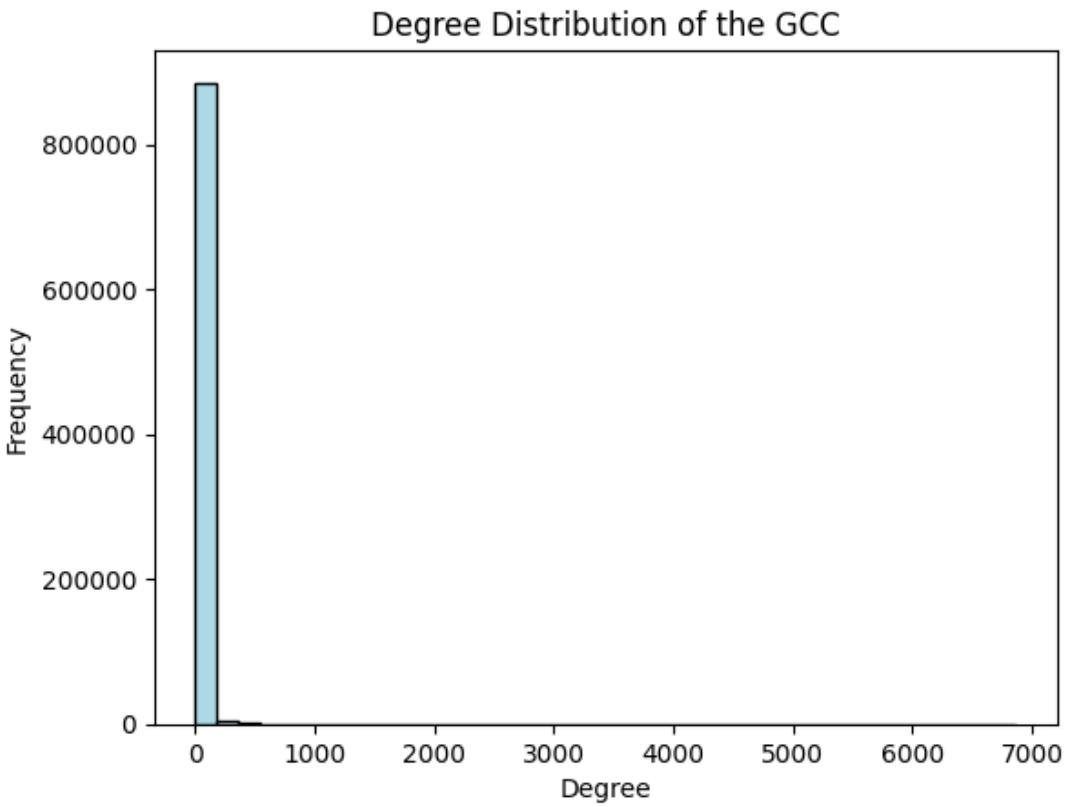
```
Number of edges filtered away is 7382
```

We can see that even though we filtered more than half of the edges away by only looking at the greatest component, we kept far most of the edges. From that, we know that many of the users with little to know friends were thinly connected together in small subgraphs.

[ ]:
```
# degree stats
degrees_GC = [G.degree(node) for node in G.nodes()]
print(f'Average degree: {sum(degrees_GC)/G.number_of_nodes()}')
print(f'Highest degree: {max(degrees_GC)}')
print(f'Lowest degree: {min(degrees_GC)}')
```

```
Average degree: 16.361543772810016
Highest degree: 6868
Lowest degree: 1
```

[ ]:
```
# degree distribution in a histogram
plt.hist(degrees_GC, bins=38, color='lightblue', edgecolor='black')
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('Degree Distribution of the GCC')
plt.show()
```



Highly skewed with many friends having very few friends

Although we have drastically cut down on the graph by only keeping the greatest component, it is still way too large to plot, thus we take 5000 random nodes and plot them just to get some visual

[ ]:
```
# select 30000 random nodes
"""
random.seed(SEED)
sampled_nodes = random.sample(list(G.nodes()), 30000)
H = G.subgraph(sampled_nodes).copy()

with open('data/processed/yelp_graph_H.pkl', 'wb') as f:
    pickle.dump(H, f)
print(" Saved graph H")
"""
```

```
Saved graph H
```

[67]:
```
H = pickle.load(open('data/processed/yelp_graph_H.pkl', 'rb'))
```

[56]:
```
number_of_nodes_full_network = H.number_of_nodes()
number_of_edges_full_network = H.number_of_edges()
```

```
print(f'Number of nodes in the graph: {number_of_nodes_full_network}')
print(f'Number of edges in the graph: {number_of_edges_full_network}')
```

```
Number of nodes in the graph: 30000
Number of edges in the graph: 8551
```

[57]:
```
# taking the greatest component
Hcc = sorted(nx.connected_components(G=H), key=len, reverse=True)
H = H.subgraph(Hcc[0])
```

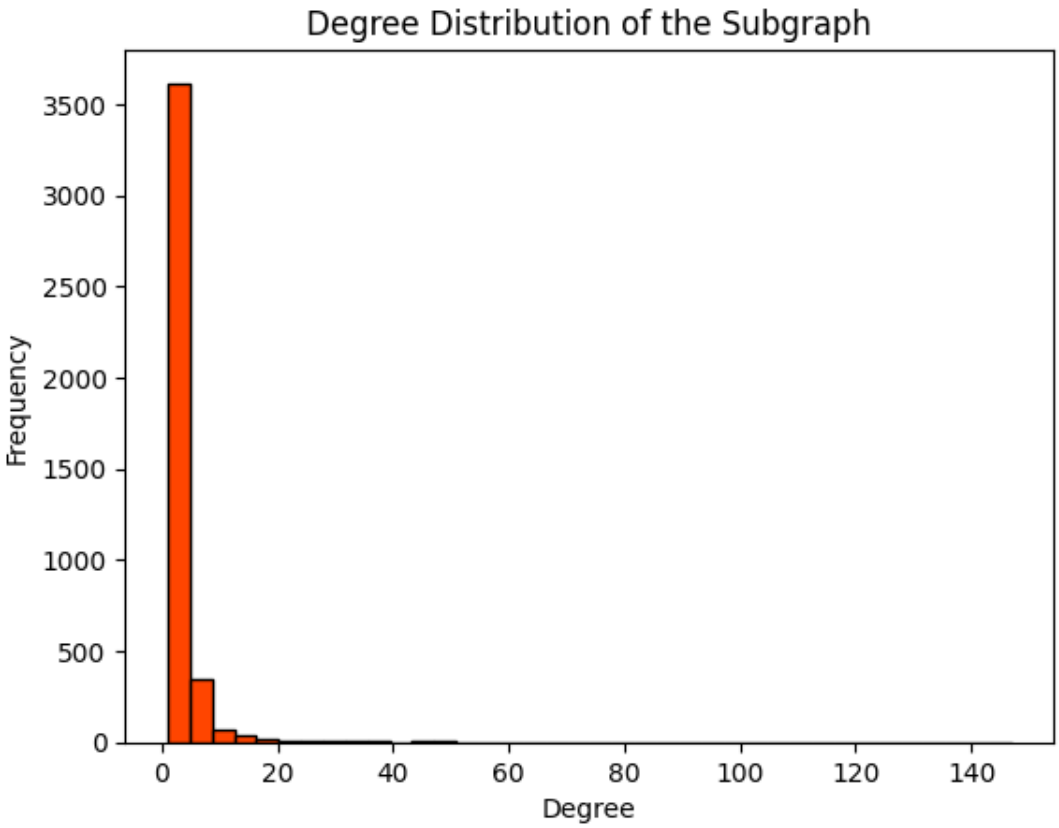
[58]:
```
number_of_nodes_full_network = H.number_of_nodes()
number_of_edges_full_network = H.number_of_edges()
print(f'Number of nodes in the graph: {number_of_nodes_full_network}')
print(f'Number of edges in the graph: {number_of_edges_full_network}')
```

```
Number of nodes in the graph: 4132
Number of edges in the graph: 6003
```

[59]:
```
degrees_HC = [H.degree(node) for node in H.nodes()]
print(f'Average degree: {sum(degrees_HC)/H.number_of_nodes()}')
print(f'Highest degree: {max(degrees_HC)}')
print(f'Lowest degree: {min(degrees_HC)}')
```

```
Average degree: 2.905614714424008
Highest degree: 147
Lowest degree: 1
```

[ ]:
```
# degree distribution in a histogram
plt.hist(degrees_HC, bins=38, color='orangered', edgecolor='black')
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('Degree Distribution of the Subgraph')
plt.savefig("figures/yelp_user_friendship_degree_distribution_subgraph.svg",␣
 ↪dpi=300)
plt.show()
```



Highly skewed degree distribution, with many nodes not having many friends. Just as with the full network.

[25]:
```
plt.figure(figsize=(10, 10))
nx.draw(H, node_size=10, alpha=0.6, edge_color="gray", with_labels=False)
plt.title("Graph of Sampled Yelp Users and Their Friendships")
plt.savefig("figures/yelp_user_friendship_graph.svg", dpi=300)
plt.show()
```

[ ]:
```
# making the graph to dict to work with it
graph_dict = nx.to_dict_of_lists(H)
```

[87]:
```
"""
sorted_betweenness = sorted(betweenness_centrality_normalized(graph_dict).
 ↪items(), key=lambda x: x[1], reverse=True)
print("Top 10 users by betweenness centrality:")
for user, centrality in sorted_betweenness[:10]:
    print(f'User ID: {user}, Betweenness Centrality: {centrality}')

sorted_betweenness = dict(sorted_betweenness)
"""
```

```
Top 10 users by betweenness centrality:
User ID: djxnI8Ux8ZYQJhiOQkrRhA, Betweenness Centrality: 0.17624815059183707
User ID: uIjj7EIVBU4kGNgmKPO02A, Betweenness Centrality: 0.1453045998841232
User ID: r3ov7FgibBx41_W74I1KiA, Betweenness Centrality: 0.08741732950571851
User ID: INxvh4Rixsdfzh6PcWc_pw, Betweenness Centrality: 0.07375305092937708
User ID: SRd7-3R8jG_WHAS5C4BfaQ, Betweenness Centrality: 0.06530469808859558
User ID: TL-wgAhbdROaC4b-DA8-7Q, Betweenness Centrality: 0.054233384866428425
User ID: oygdh1nR-FyWsqP7ajoTvw, Betweenness Centrality: 0.05337731914642875
User ID: tJIzUxwfaLDwO6R6wSZ9vw, Betweenness Centrality: 0.05277453588220559
User ID: jt49xjEjQisu6wTTGn6B3A, Betweenness Centrality: 0.05197145358820639
User ID: C6LVOp8L6IfBfh1YFRkWzQ, Betweenness Centrality: 0.051165107178772484
```

[ ]:
```
# save to csv to be able to recreate
"""
betweenness_df = pd.DataFrame.from_dict(sorted_betweenness, orient='index',␣
 ↪columns=['betweenness_centrality'])
betweenness_df.index.name = 'user_id'
betweenness_df.to_csv('data/processed/yelp_betweenness_centrality.csv')
"""
```

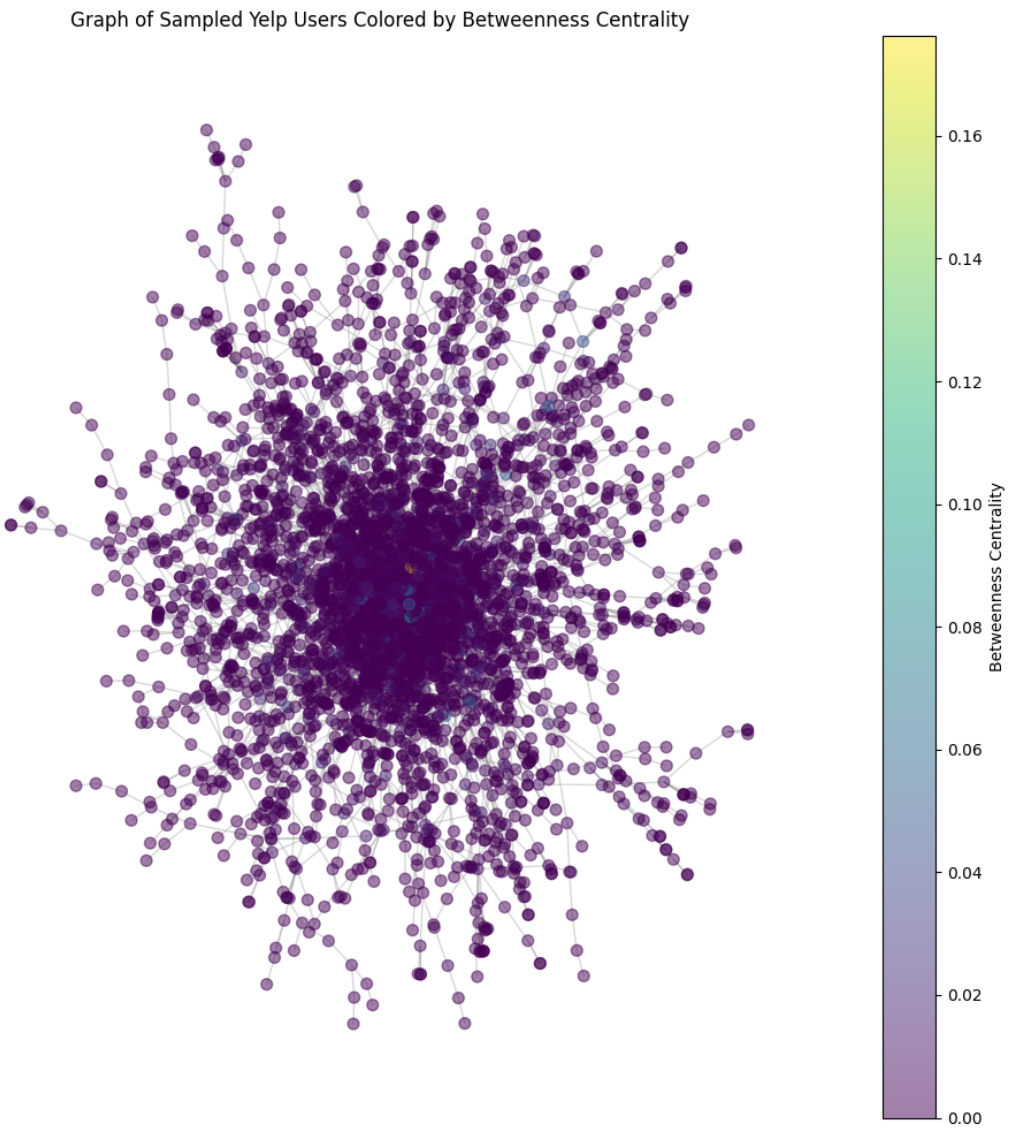
[62]:
```
betweenness_df = pd.read_csv('data/processed/yelp_betweenness_centrality.csv',␣
 ↪index_col='user_id')
sorted_betweenness = betweenness_df['betweenness_centrality'].to_dict()
```

[29]:
```
# Comparing to the built-in function
"""
betweenness_centrality_built_in = nx.betweenness_centrality(H)
sorted_betweenness_built_in = sorted(betweenness_centrality_built_in.items(),␣
 ↪key=lambda x: x[1], reverse=True)
print("Top 10 users by betweenness centrality (built-in):")
for user, centrality in sorted_betweenness_built_in[:10]:
    print(f'User ID: {user}, Betweenness Centrality: {centrality}')
"""
```

[29]:
```
'\nbetweenness_centrality_built_in =
nx.betweenness_centrality(H)\nsorted_betweenness_built_in =
sorted(betweenness_centrality_built_in.items(), key=lambda x: x[1],
reverse=True)\nprint("Top 10 users by betweenness centrality (built-in):")\nfor
user, centrality in sorted_betweenness_built_in[:10]:\n    print(f\'User ID:
{user}, Betweenness Centrality: {centrality}\')\n'
```

[ ]:
```
# Change to the manual calculation
node_colors = [sorted_betweenness[n] for n in H.nodes()]

pos = nx.spring_layout(H, seed=SEED)

fig, ax = plt.subplots(figsize=(10, 10))
```

```
nodes = nx.draw_networkx_nodes(
    H, pos, ax=ax,
    node_size=50,
    node_color=node_colors,
    cmap=plt.cm.viridis,
    alpha=0.5
)
nx.draw_networkx_edges(H, pos, ax=ax, edge_color="gray", alpha=0.3)

ax.set_title("Graph of Sampled Yelp Users Colored by Betweenness Centrality")
ax.set_axis_off()

cbar = fig.colorbar(nodes, ax=ax)
cbar.set_label("Betweenness Centrality")

plt.tight_layout()
plt.savefig("figures/yelp_user_friendship_betweenness_centrality.svg", dpi=300)
plt.show()
```



The nodes with highest betweenness is in the middle which does make sense as they are most likely on most of the shortest paths.

[ ]:
```
"""
mods = {}
for k in range(30, 50):
    labels = spectral_clustering(graph_dict, k)
    communities = {}
    for node, lbl in labels.items():
        communities.setdefault(lbl, []).append(node)
    mods[k] = modularity(H, communities.values())
```

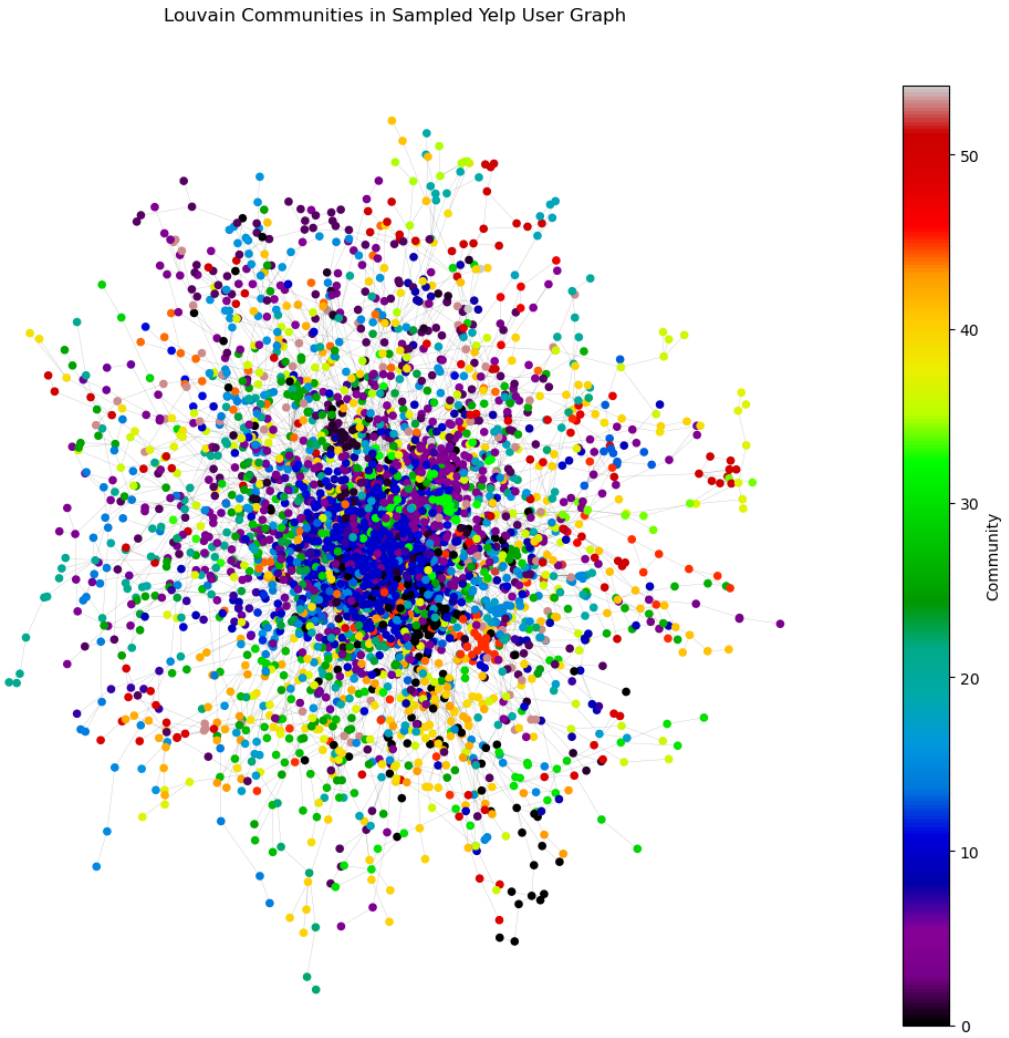
```
print(mods)
print("Optimal number of clusters (k) based on highest modularity:", max(mods,␣
↪key=mods.get))
"""
```

```
{30: 0.35784138859550174, 31: 0.3140518696174152, 32: 0.35870140336572787, 33:
0.3599850499537838, 34: 0.36667877122574805, 35: 0.3466155339233043, 36:
0.35159757841108324, 37: 0.3451614050823443, 38: 0.3735820467799306, 39:
0.37540314189620727, 40: 0.38129667744283224, 41: 0.3457213866274704, 42:
0.37888744838530813, 43: 0.35285398835370485, 44: 0.39345018756100325, 45:
0.3695041118454599, 46: 0.38573416939706073, 47: 0.39811300968428537, 48:
0.41849287472427926, 49: 0.3942257590178756}
Optimal number of clusters (k) based on highest modularity: 48
```

[65]:
```
# pickle the spectral communities
"""
with open('data/processed/yelp_spectral_communities.pkl', 'wb') as f:
    pickle.dump(communities, f)"""

with open('data/processed/yelp_spectral_communities.pkl', 'rb') as f:
    communities = pickle.load(f)
```

[ ]:
```
"""# Louvain partition
partition = community_louvain.best_partition(H, random_state=SEED,␣
↪randomize=False)

# Map community labels
com_ids = {c: i for i, c in enumerate(sorted(set(partition.values())))}
node_colors = [com_ids[partition[n]] for n in H.nodes()]
num_comms = len(com_ids)

pos = nx.spring_layout(H, seed=SEED)

fig, ax = plt.subplots(figsize=(10, 10))
nodes = nx.draw_networkx_nodes(
    H, pos, ax=ax,
    nodelist=list(H.nodes()),
    node_size=20,
    node_color=node_colors,
    cmap=plt.cm.tab20 if num_comms <= 20 else plt.cm.viridis
)
nx.draw_networkx_edges(H, pos, ax=ax, alpha=0.3, width=0.2)

# Colormap for better distinction between communities
nodes.set_cmap(plt.cm.tab20b if num_comms <= 20 else plt.cm.nipy_spectral)
ax.set_title("Louvain Communities in Sampled Yelp User Graph")
ax.set_axis_off()
```

```
cbar = fig.colorbar(nodes, ax=ax, fraction=0.046, pad=0.04)
cbar.set_label("Community")
if num_comms <= 20:
    cbar.set_ticks(range(num_comms))
    cbar.set_ticklabels(range(num_comms))

plt.tight_layout()
plt.savefig("figures/yelp_user_friendship_louvain_communities.svg", dpi=300)
plt.show()
"""
```



In above we see all the communities in one graph. It is however really hard to tell anything from

this.

[ ]:
```
"""
sizes = pd.Series(list(partition.values())).value_counts()
# number of communities
num_communities = len(sizes)
print(f'Number of communities detected by Louvain method: {num_communities}')
comms = {}
for n,c in partition.items():
    comms.setdefault(c, []).append(n)

modularity_score = modularity(H, comms.values())
print(modularity_score)
"""
```

```
Number of communities detected by Louvain method: 55
0.7691229625345026
```

Better modularity for Louvain communities, thus we want to work with this

[62]:
```
# pickle the communities
"""
with open('data/processed/yelp_louvain_communities.pkl', 'wb') as f:
    pickle.dump(partition, f)
"""

with open('data/processed/yelp_louvain_communities.pkl', 'rb') as f:
    partition = pickle.load(f)
```

[63]:
```
# user_ids in community 6
users_in_community_6 = [node for node, comm in partition.items() if comm == 6]
print(f'User IDs in community 6: {users_in_community_6}')
```

```
User IDs in community 6: ['jPBKGmxgfG5iRipd3v7BAw', 'wuT2zH7Atq-98BirpB-8Aw',
'5OZvqngWSRvdMeQQRkwzLw', 'UOAq6-YRLoIgWtJNkChzbA', 'V9doxrjqG6S94V9WpHGr7w',
'bbxczHvKDVJbx8G6KyeCaQ', 'h5kN9PDhS3XJ7toa3PVJ9Q', 'mcXs_mV5CQHhlUZsRVvtXA',
'LhZLS33umZG20LfyEftYeQ', 'UGTQum1peLSeOZr08LERbA', '4-Oh0TN8-b9gj12_VUZmAw',
'pZioOLzVNpxyEVchL6LLsQ', '2x4SFLumawQPPzYcx6v_7Q', 'yhyjGxcZ3ypdg_470CCgKw',
'LOK4oCby0Ou46W65YMO6-w', 'R5r52e3EnFDStcztaN_eFg', '1eQcZZWwGMv45rkjRepNaA',
'cuIGSOIcsYdhcf16oy7RiQ', 'GX_lTNelZdEDx2XF8ECrQQ', 'q3zGja1DWAr5X3FRzLLrrQ',
'w84Uv36jWR5bZwEUUIH_2A', 'uOI4jHuwN-XJvd6zkMBtRQ', 'B9wxFyA_Fkr4b2GTdX3uNw',
'OSssus6Lt58zVRdcB-nlWg', 'sAbHdUgrjMOiQQLNh0H0YQ', 'ZJ6sj1IjdwmPPL_ZxmRKgw',
'ZOjaV7oALZrBMznTxhfIWA', 'F4Abg96un5mT019hdstmVA', 'VzPkbXNATe_7f5CLCtqyPA',
'c_YjHxlnDARw6UothG05sg', '8OVF-4kKZZNMXX_vuYiPQw', 'npaqVmBLp_bRk4DcraZYww',
'HMUnp55Q8_vxIEPl1V0W_g', 'Sm8Vc5ebaw4CRooUKlvv6Q', 'qv9Ke7SBcg4j8lHkcPK2LA',
'67UFw_MqwDC_XF5bwhqF5Q', 'vl-GujIYlPJfC-fHz-5xVg', '1VdMQhgS18-vc1oY_Lf6xw',
'Q83R2UssCoErEJX36wrlvg', '4BaRBsPhITxM8GhMFhKFnw', 'zxxtMBaLWnhFc6ZBDDsCtg',
'hYPVPwbAdcaPeqJ2N_PAcA', 'u6FIRvx5x5ry4UHbRz9nEg', '-uaP5ca_Z2aJTIW7n4bmcQ',
'DNmGVmGPvwr88suI7P3lCQ', 'CXplwx9fHqJ6n6FXmMx3FQ', 'm6w01BJpTJlROqa5xCtJzw',
```

```
'k-937AdLZ-ITxLCFNIrBJg']
```

[64]:
```
sizes = pd.Series(list(partition.values())).value_counts()
print(sizes)
```

```
10    633
5     224
0     207
4     203
3     190
2     177
15    159
9     157
25    139
40    107
1     107
36    100
37     88
16     87
32     82
19     76
39     75
53     75
7      72
8      70
45     65
21     65
41     63
27     63
23     60
51     57
14     57
29     55
42     51
13     51
26     51
44     50
6      48
11     48
17     41
18     38
22     38
43     36
49     28
20     26
30     21
48     17
34     14
```

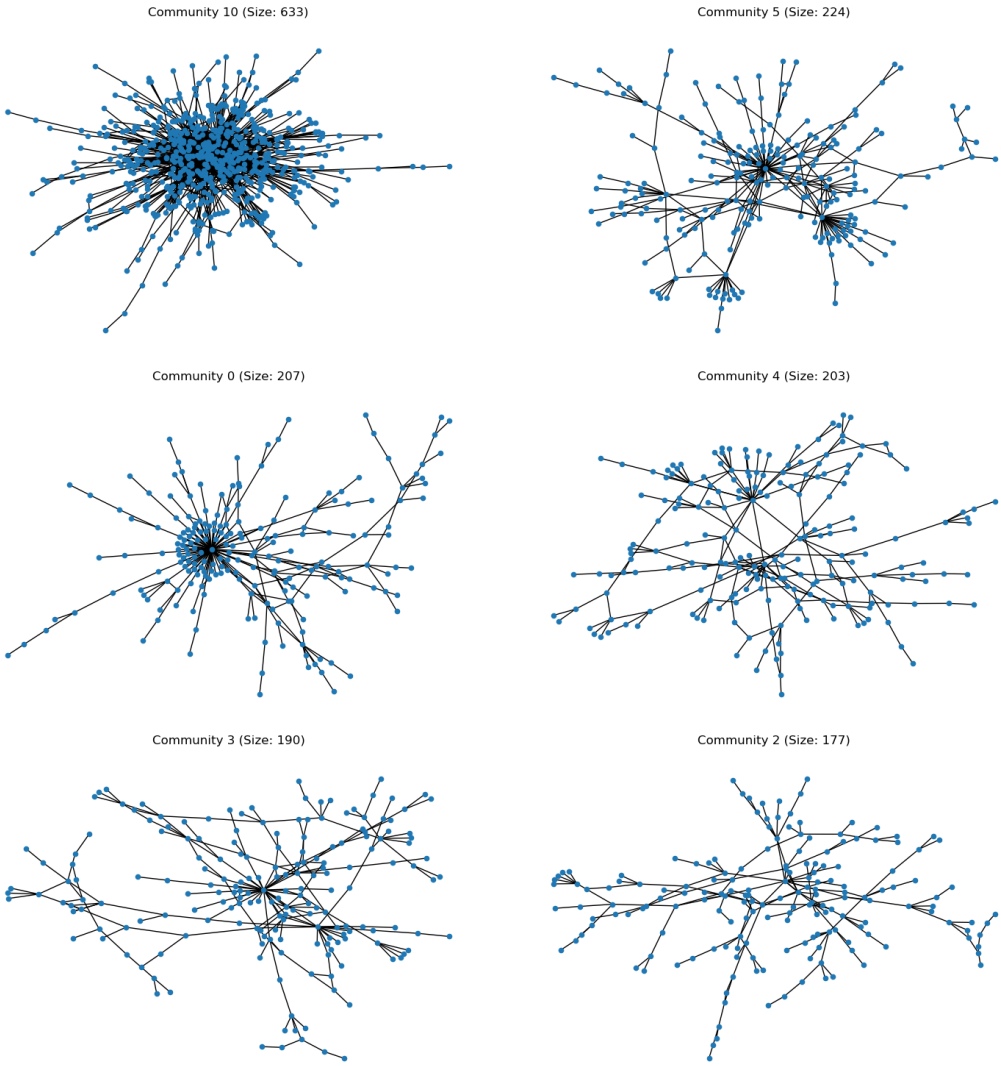
```
35     10
50      7
28      6
47      6
12      5
54      5
38      4
46      4
31      4
33      4
52      3
24      3
Name: count, dtype: int64
```

[65]:
```python
# user_ids in community 6
users_in_community_6 = [node for node, comm in partition.items() if comm == 6]
print(f'User IDs in community 6: {users_in_community_6}')
```

```
User IDs in community 6: ['jPBKGmxgfG5iRipd3v7BAw', 'wuT2zH7Atq-98BirpB-8Aw',
'5OZvqngWSRvdMeQQRkwzLw', 'UOAq6-YRLoIgWtJNkChzbA', 'V9doxrjqG6S94V9WpHGr7w',
'bbxczHvKDVJbx8G6KyeCaQ', 'h5kN9PDhS3XJ7toa3PVJ9Q', 'mcXs_mV5CQHhlUZsRVvtXA',
'LhZLS33umZG20LfyEftYeQ', 'UGTQum1peLSeOZr08LERbA', '4-Oh0TN8-b9gj12_VUZmAw',
'pZioOLzVNpxyEVchL6LLsQ', '2x4SFLumawQPPzYcx6v_7Q', 'yhyjGxcZ3ypdg_470CCgKw',
'LOK4oCby0Ou46W65YMO6-w', 'R5r52e3EnFDStcztaN_eFg', '1eQcZZWwGMv45rkjRepNaA',
'cuIGSOIcsYdhcf16oy7RiQ', 'GX_lTNelZdEDx2XF8ECrQQ', 'q3zGja1DWAr5X3FRzLLrrQ',
'w84Uv36jWR5bZwEUUIH_2A', 'uOI4jHuwN-XJvd6zkMBtRQ', 'B9wxFyA_Fkr4b2GTdX3uNw',
'OSssus6Lt58zVRdcB-nlWg', 'sAbHdUgrjMOiQQLNh0HOYQ', 'ZJ6sj1IjdwmPPL_ZxmRKgw',
'ZOjaV7oALZrBMznTxhfIWA', 'F4Abg96un5mT019hdstmVA', 'VzPkbXNATe_7f5CLCtqyPA',
'c_YjHxlnDARw6UothG05sg', '8OVF-4kKZZNMXX_vuYiPQw', 'npaqVmBLp_bRk4DcraZYww',
'HMUnp55Q8_vxIEPl1VOW_g', 'Sm8Vc5ebaw4CRooUKlvv6Q', 'qv9Ke7SBcg4j8lHkcPK2LA',
'67UFw_MqwDC_XF5bwhqF5Q', 'vl-GujIYlPJfC-fHz-5xVg', '1VdMQhgS18-vc1oY_Lf6xw',
'Q83R2UssCoErEJX36wrlvg', '4BaRBsPhITxM8GhMFhKFnw', 'zxxtMBaLWnhFc6ZBDDsCtg',
'hYPVPwbAdcaPeqJ2N_PAcA', 'u6FIRvx5x5ry4UHbRz9nEg', '-uaP5ca_Z2aJTIW7n4bmcQ',
'DNmGVmGPvwr88suI7P3lCQ', 'CXplwx9fHqJ6n6FXmMx3FQ', 'm6w01BJpTJlROqa5xCtJzw',
'k-937AdLZ-ITxLCFNIrBJg']
```

[68]:
```python
top_6_communities = sizes.index[:6]

plt.figure(figsize=(15, 15))
for i, community in enumerate(top_6_communities, 1):
    nodes = [n for n, c in partition.items() if c == community]
    sub = H.subgraph(nodes)
    plt.subplot(3, 2, i)
    nx.draw(sub, with_labels=False, node_size=20)
    plt.title(f'Community {community} (Size: {sizes[community]})')

plt.tight_layout()
plt.savefig("figures/yelp_user_friendship_top6_communities.svg", dpi=300)
```

```
plt.show()
```



[69]:
```
p = Path("data/processed/yelp_louvain_communities.pkl")
print("exists:", p.exists())
print("size:", p.stat().st_size if p.exists() else None)
```

```
exists: True
size: 111597
```

[70]:
```
user_community_df = pd.DataFrame(
    [(node, comm) for node, comm in partition.items()],
    columns=["user_id", "community"]
)
```

```
# save as csv
user_community_df.to_csv(DATA_DIR_PROC / "yelp_user_communities.csv",␣
 ↪index=False)
```

7 Apriori on the communities

Now, we run the apriori algorithm on each of the communities found in the above seciton. We start by collecting the set of users in each community.

[71]:
```
# size of the communities
sizes = pd.Series(list(partition.values())).value_counts()

# Top 6 largest communities
top_6_communities = sizes.index[:6].tolist()
print(f"Top 6 communities (by size): {top_6_communities}")
print(f"Sizes: {sizes[top_6_communities].tolist()}")
```

```
Top 6 communities (by size): [10, 5, 0, 4, 3, 2]
Sizes: [633, 224, 207, 203, 190, 177]
```

[96]:
```
# Create a mapping: user_id -> community_id
user_to_community = partition

# Get user sets for each community
top_6_users = {
    comm: set([user for user, c in partition.items() if c == comm])
    for comm in top_6_communities
}
```

Now, we keep only baskets of the users in the identified communities.

[ ]:
```
# Load original baskets
liked_baskets = pd.read_parquet("data/processed/baskets_liked_ids.parquet")
hated_baskets = pd.read_parquet("data/processed/baskets_hated_ids.parquet")

# Filter for each community
communities_to_analyze = {
    **{f"top_{i+1}": (comm, top_6_users[comm]) for i, comm in␣
 ↪enumerate(top_6_communities)}
}

# filter baskets
filtered_data = {}
for name, (comm_id, user_set) in communities_to_analyze.items():
    filtered_data[name] = {
        'community_id': comm_id,
        'liked': ap.filter_baskets_by_community(liked_baskets, user_set),
```

```
        'hated': ap.filter_baskets_by_community(hated_baskets, user_set)
    }
    print(f"{name} (comm {comm_id}): {len(filtered_data[name]['liked'])} liked␣
 ↪baskets, {len(filtered_data[name]['hated'])} hated baskets")
```

```
top_1 (comm 10): 251 liked baskets, 58 hated baskets
top_2 (comm 5): 98 liked baskets, 33 hated baskets
top_3 (comm 0): 96 liked baskets, 34 hated baskets
top_4 (comm 4): 62 liked baskets, 24 hated baskets
top_5 (comm 3): 70 liked baskets, 18 hated baskets
top_6 (comm 2): 43 liked baskets, 7 hated baskets
```

Since each community is different, we need to select community specific thresholds for support and confidence. These numbers are select based on trial and error to get a reasonable number of rules per community.

[ ]:
```
COMM_DIR = DATA_DIR_PROC / "baskets_by_community"
COMM_DIR.mkdir(parents=True, exist_ok=True)

# Define community-specific thresholds
COMM_THRESHOLDS = {
    "top_1": {"min_support": 0.06, "min_conf": 0.6},
    "top_2": {"min_support": 0.1, "min_conf": 0.6},
    "top_3": {"min_support": 0.1, "min_conf": 0.6},
    "top_4": {"min_support": 0.196, "min_conf": 0.6},
    "top_5": {"min_support": 0.066, "min_conf": 0.6},
    "top_6": {"min_support": 0.066, "min_conf": 0.6},
}

cols_decoded = ["Antecedent", "Consequent", "Antecedent_decoded",␣
 ↪"Consequent_decoded",
                "Support", "Confidence", "Lift"]

for name, (comm_id, user_set) in communities_to_analyze.items():
    print(f"\nCommunity {name} (ID {comm_id})")

    liked_comm = ap.filter_baskets_by_community(liked_baskets, user_set)
    hated_comm = ap.filter_baskets_by_community(hated_baskets, user_set)

    liked_path = COMM_DIR / f"baskets_liked_{name}_comm{comm_id}.parquet"
    hated_path = COMM_DIR / f"baskets_hated_{name}_comm{comm_id}.parquet"

    liked_comm.to_parquet(liked_path, index=False)
    hated_comm.to_parquet(hated_path, index=False)

    threashold = COMM_THRESHOLDS.get(name, {"min_support": 0.02, "min_conf": 0.
 ↪4})
    MIN_SUPPORT = threashold["min_support"]
```

```
    MIN_CONF    = threashold["min_conf"]

    df_rules_liked_comm, _ = ap.run_apriori(
        f"{name}_liked", liked_path, MIN_SUPPORT, MIN_CONF, OUT_DIR, id2cat_map
    )
    df_rules_hated_comm, _ = ap.run_apriori(
        f"{name}_hated", hated_path, MIN_SUPPORT, MIN_CONF, OUT_DIR, id2cat_map
    )
```

```
Community top_1 (ID 10)
Running Apriori for top_1_liked
Generated 258 top_1_liked rules
Saved 258 top_1_liked rules
Running Apriori for top_1_hated
Generated 1286 top_1_hated rules
Saved 1286 top_1_hated rules

Community top_2 (ID 5)
Running Apriori for top_2_liked
Generated 1206 top_2_liked rules
Saved 1206 top_2_liked rules
Running Apriori for top_2_hated
Generated 3 top_2_hated rules
Saved 3 top_2_hated rules

Community top_3 (ID 0)
Running Apriori for top_3_liked
Generated 986 top_3_liked rules
Saved 986 top_3_liked rules
Running Apriori for top_3_hated
Generated 10 top_3_hated rules
Saved 10 top_3_hated rules

Community top_4 (ID 4)
Running Apriori for top_4_liked
Generated 45 top_4_liked rules
Saved 45 top_4_liked rules
Running Apriori for top_4_hated
Generated 3 top_4_hated rules
Saved 3 top_4_hated rules

Community top_5 (ID 3)
Running Apriori for top_5_liked
Generated 333 top_5_liked rules
Saved 333 top_5_liked rules
Running Apriori for top_5_hated
Generated 1107 top_5_hated rules
```

```
Saved 1107 top_5_hated rules

Community top_6 (ID 2)
Running Apriori for top_6_liked
Generated 4 top_6_liked rules
Saved 4 top_6_liked rules
Running Apriori for top_6_hated
Generated 6224 top_6_hated rules
Saved 6224 top_6_hated rules
```

8 We load in all the rules found for each community and filter them based on the values of support, confidence and lift same as we done for the global rules, to get a comparable results.

[ ]:
```
# Load community rules from parquet files to preserve tuple data types
rules_top1_liked = pd.read_parquet(OUT_DIR / "rules_top_1_liked.parquet")
rules_top1_hated = pd.read_parquet(OUT_DIR / "rules_top_1_hated.parquet")
rules_top2_liked = pd.read_parquet(OUT_DIR / "rules_top_2_liked.parquet")
rules_top2_hated = pd.read_parquet(OUT_DIR / "rules_top_2_hated.parquet")
rules_top3_liked = pd.read_parquet(OUT_DIR / "rules_top_3_liked.parquet")
rules_top3_hated = pd.read_parquet(OUT_DIR / "rules_top_3_hated.parquet")
rules_top4_liked = pd.read_parquet(OUT_DIR / "rules_top_4_liked.parquet")
rules_top4_hated = pd.read_parquet(OUT_DIR / "rules_top_4_hated.parquet")
rules_top5_liked = pd.read_parquet(OUT_DIR / "rules_top_5_liked.parquet")
rules_top5_hated = pd.read_parquet(OUT_DIR / "rules_top_5_hated.parquet")
rules_top6_liked = pd.read_parquet(OUT_DIR / "rules_top_6_liked.parquet")
rules_top6_hated = pd.read_parquet(OUT_DIR / "rules_top_6_hated.parquet")

# Filter out the rules based on the global values
rules_top_1_filtered_liked = ap.filter_rules(rules_top1_liked, min_support=0.
 ↪02, min_confidence=0.5, min_lift=2, max_antecedent=3, color="Greens",␣
 ↪save_path="figures/rules_top1_liked_filtered.html")
rules_top_1_filtred_hated = ap.filter_rules(rules_top1_hated, min_support=0.
 ↪012, min_confidence=0.4, min_lift=1.6, max_antecedent=3, color="Reds",␣
 ↪save_path="figures/rules_top1_hated_filtered.html")
rules_top_2_filtered_liked = ap.filter_rules(rules_top2_liked, min_support=0.
 ↪02, min_confidence=0.5, min_lift=2, max_antecedent=3, color="Greens",␣
 ↪save_path="figures/rules_top2_liked_filtered.html")
rules_top_2_filtred_hated = ap.filter_rules(rules_top2_hated, min_support=0.
 ↪012, min_confidence=0.4, min_lift=1.6, max_antecedent=3, color="Reds",␣
 ↪save_path="figures/rules_top2_hated_filtered.html")
rules_top_3_filtered_liked = ap.filter_rules(rules_top3_liked, min_support=0.
 ↪02, min_confidence=0.5, min_lift=2, max_antecedent=3, color="Greens",␣
 ↪save_path="figures/rules_top3_liked_filtered.html")
```

```
rules_top_3_filtred_hated = ap.filter_rules(rules_top3_hated, min_support=0.
  ↪012, min_confidence=0.4, min_lift=1.6, max_antecedent=3, color="Reds",␣
  ↪save_path="figures/rules_top3_hated_filtered.html")
rules_top_4_filtered_liked = ap.filter_rules(rules_top4_liked, min_support=0.
  ↪02, min_confidence=0.5, min_lift=2, max_antecedent=3, color="Greens",␣
  ↪save_path="figures/rules_top4_liked_filtered.html")
rules_top_4_filtred_hated = ap.filter_rules(rules_top4_hated, min_support=0.
  ↪012, min_confidence=0.4, min_lift=1.6, max_antecedent=3, color="Reds",␣
  ↪save_path="figures/rules_top4_hated_filtered.html")
rules_top_5_filtered_liked = ap.filter_rules(rules_top5_liked, min_support=0.
  ↪02, min_confidence=0.5, min_lift=2, max_antecedent=3, color="Greens",␣
  ↪save_path="figures/rules_top5_liked_filtered.html")
rules_top_5_filtred_hated = ap.filter_rules(rules_top5_hated, min_support=0.
  ↪012, min_confidence=0.4, min_lift=1.6, max_antecedent=3, color="Reds",␣
  ↪save_path="figures/rules_top5_hated_filtered.html")
rules_top_6_filtered_liked = ap.filter_rules(rules_top6_liked, min_support=0.
  ↪02, min_confidence=0.5, min_lift=2, max_antecedent=3, color="Greens",␣
  ↪save_path="figures/rules_top6_liked_filtered.html")
rules_top_6_filtred_hated = ap.filter_rules(rules_top6_hated, min_support=0.
  ↪012, min_confidence=0.4, min_lift=1.6, max_antecedent=3, color="Reds",␣
  ↪save_path="figures/rules_top6_hated_filtered.html")
```

8.1 Investigating the rules for communities

8.1.1 Community 1: Liked categories and hated categories

[110]:
```
rules_top1_liked_lift = (
    rules_top1_liked
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top1_liked_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Greens, subset=num_cols)
```

[110]:
```
<pandas.io.formats.style.Styler at 0x7f40190e9e50>
```

[111]:
```
rules_top1_hated_lift = (
    rules_top1_hated
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top1_hated_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Reds, subset=num_cols)
```

[111]: <pandas.io.formats.style.Styler at 0x7f3c38fdc310>

8.1.2 Community 2: Liked categories and hated categories

[112]:
```
rules_top2_liked_lift = (
    rules_top2_liked
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top2_liked_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Greens, subset=num_cols)
```

[112]: <pandas.io.formats.style.Styler at 0x7f3c38fde190>

[113]:
```
rules_top2_hated_lift = (
    rules_top2_hated
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top2_hated_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Reds, subset=num_cols)
```

[113]: <pandas.io.formats.style.Styler at 0x7f3c38fdc350>

8.1.3 Community 3 : Liked categories and hated categories

[114]:
```
rules_top3_liked_lift = (
    rules_top3_liked
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top3_liked_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Greens, subset=num_cols)
```

[114]: <pandas.io.formats.style.Styler at 0x7f3c38ff1e50>

[115]:
```
rules_top3_hated_lift = (
    rules_top3_hated
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
```

```
num_cols = ["Support", "Confidence", "Lift"]
rules_top3_hated_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Reds, subset=num_cols)
```

[115]: <pandas.io.formats.style.Styler at 0x7f3c38fd35d0>

8.1.4 Community 4: Liked categories and hated categories

[116]:
```
rules_top4_liked_lift = (
    rules_top4_liked
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top4_liked_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Greens, subset=num_cols)
```

[116]: <pandas.io.formats.style.Styler at 0x7f3c38ff7450>

[117]:
```
rules_top4_hated_lift = (
    rules_top4_hated
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top4_hated_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Reds, subset=num_cols)
```

[117]: <pandas.io.formats.style.Styler at 0x7f3c38fd0510>

8.1.5 Community 5: Liked categories and hated categories

[118]:
```
rules_top5_liked_lift = (
    rules_top5_liked
    .sort_values(["Lift"], ascending=False)
    .head()
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top5_liked_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Greens, subset=num_cols)
```

[118]: <pandas.io.formats.style.Styler at 0x7f3c38e04850>

[119]:
```
rules_top5_hated_lift = (
    rules_top5_hated
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top5_hated_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Reds, subset=num_cols)
```

[119]:
```
<pandas.io.formats.style.Styler at 0x7f3c38ff8ad0>
```

8.1.6 Community 6: Liked categories and hated categories

[120]:
```
rules_top6_liked_lift = (
    rules_top6_liked
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top6_liked_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Greens, subset=num_cols)
```

[120]:
```
<pandas.io.formats.style.Styler at 0x7f3c38ff10d0>
```

[ ]:
```
rules_top6_hated_lift = (
    rules_top6_hated
    .sort_values(["Lift"], ascending=False)
    .head(3)
)
num_cols = ["Support", "Confidence", "Lift"]
rules_top6_hated_lift.style \
    .format({c: "{:.4f}" for c in num_cols}) \
    .background_gradient(cmap=cm.Reds, subset=num_cols)
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x7f40185f5990>
```

Now we calculate the jaccard similarity between the global rules and the community rules to see if there are any overlaps.

[123]:
```
liked_filtered = [
    rules_liked_global,
    rules_top_1_filtered_liked,
    rules_top_2_filtered_liked,
    rules_top_3_filtered_liked,
    rules_top_4_filtered_liked,
    rules_top_5_filtered_liked,
```

```
    rules_top_6_filtered_liked,
]

jaccard_liked = []

for k, df_comm in enumerate(liked_filtered):
    sim = ap.jaccard_similariy(rules_liked_global, df_comm)
    jaccard_liked.append(sim)
    print(f"Jaccard (liked global vs top_{k} liked): {sim:.4f}")
```

```
Jaccard (liked global vs top_0 liked): 1.0000
Jaccard (liked global vs top_1 liked): 0.1197
Jaccard (liked global vs top_2 liked): 0.0281
Jaccard (liked global vs top_3 liked): 0.0208
Jaccard (liked global vs top_4 liked): 0.0109
Jaccard (liked global vs top_5 liked): 0.0402
Jaccard (liked global vs top_6 liked): 0.0000
```

We see very minimal overlap between the global rules and the community rules.

Now, we calcualte the average values of support, confidence and lift for the rules found in each community and compare them to the global rules.

[124]:
```
# Get mean row for each liked rule set
liked_means = [df.describe().loc["mean"] for df in liked_filtered]

# Combine them into one DataFrame
liked_means_df = pd.DataFrame(liked_means)

# Optional: give them names for clarity
liked_means_df.index = [
    "global",
    "top_1",
    "top_2",
    "top_3",
    "top_4",
    "top_5",
    "top_6"
]

print(liked_means_df)
```

```
         Support  Confidence      Lift
global  0.024914    0.574407  2.361651
top_1   0.072200    0.763907  3.003140
top_2   0.113153    0.811910  3.415280
top_3   0.110573    0.793072  3.362916
top_4   0.211694    0.809824  2.356342
top_5   0.075719    0.792959  4.170202
```

```
top_6    0.081395     0.737500  3.660972
```

We observe, that the values are higher for the community rules compared to the global rules.

We perform the same analysis on the hated categories as well.

[ ]:
```python
hated_filtered = [
    rules_hated_global,
    rules_top_1_filtred_hated,
    rules_top_2_filtred_hated,
    rules_top_3_filtred_hated,
    rules_top_4_filtred_hated,
    rules_top_5_filtred_hated,
    rules_top_6_filtred_hated,
]

jaccard_hated = []

for k, df_comm in enumerate(hated_filtered):
    sim = ap.jaccard_similariy(rules_hated_global, df_comm)
    jaccard_hated.append(sim)
    print(f"Jaccard (hated global vs top_{k} hated): {sim:.4f}")
```

```
Jaccard (hated global vs top_0 hated): 1.0000
Jaccard (hated global vs top_1 hated): 0.0027
Jaccard (hated global vs top_2 hated): 0.0000
Jaccard (hated global vs top_3 hated): 0.0714
Jaccard (hated global vs top_4 hated): 0.0000
Jaccard (hated global vs top_5 hated): 0.0021
Jaccard (hated global vs top_6 hated): 0.0000
```

[126]:
```python
# Get mean row for each liked rule set
hated_means = [df.describe().loc["mean"] for df in hated_filtered]

# Combine them into one DataFrame
hated_means_df = pd.DataFrame(hated_means)

# Optional: give them names for clarity
hated_means_df.index = [
    "global",
    "top_1",
    "top_2",
    "top_3",
    "top_4",
    "top_5",
    "top_6"
]

print(hated_means_df)
```

	Support	Confidence	Lift
global	0.012789	0.422328	1.723500
top_1	0.072635	0.838931	7.262171
top_2	0.121212	0.866667	2.876923
top_3	0.129412	0.807857	2.895979
top_4	0.222222	0.732143	2.051948
top_5	0.114444	0.907787	5.877170
top_6	0.142857	1.000000	6.712358

We observe as well, that the values are higher for the community rules compared to the global rules.

9 Sentiment Analysis

Lastly, we use sentiment analysis to to analyze the reviews in each communites, to check how does the sentiment scores vary across the communities and if they are consistent with the liked and hated categories found in each community.

We use RoBERTa model.

[138]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

MODEL = "cardiffnlp/twitter-roberta-base-sentiment-latest"

tokenizer = AutoTokenizer.from_pretrained(MODEL)
tokenizer.model_max_length = 512

model = AutoModelForSequenceClassification.
 ↪from_pretrained(MODEL,use_safetensors=True)

model = model.to(device)

sentiment = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer,␣
 ↪device=device,)
```

```
'(ProtocolError('Connection aborted.', RemoteDisconnected('Remote end closed
connection without response')), '(Request ID:
6828d020-acb4-4eb4-820d-2ed9e8045625)')' thrown while requesting HEAD
https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-
latest/resolve/main/tokenizer_config.json
Retrying in 1s [Retry 1/5].

Some weights of the model checkpoint at cardiffnlp/twitter-roberta-base-
sentiment-latest were not used when initializing
RobertaForSequenceClassification: ['roberta.pooler.dense.bias',
'roberta.pooler.dense.weight']
- This IS expected if you are initializing RobertaForSequenceClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
```

```
BertForPreTraining model).
- This IS NOT expected if you are initializing RobertaForSequenceClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Device set to use cuda
```

Now, we make a mapping for each user to their community ID.

[132]:
```
user_to_comm = {}

for name, (comm_id, user_set) in communities_to_analyze.items():
    for u in user_set:
        user_to_comm[u] = name

target_users = set(user_to_comm.keys())
print(f"Total users in selected communities: {len(target_users)}")
```

```
Total users in selected communities: 1634
```

Now we filter out reviews for users that belongs to the identified top 6 communities. We attach the community ID to each review as well. And then we run the sentiment analysis on the reviews in each community.

[139]:
```
SENT_DIR = DATA_DIR_PROC / "sentiment"
SENT_DIR.mkdir(parents=True, exist_ok=True)

chunk_size = 50_000

all_chunks = []

usecols = ["review_id", "user_id", "business_id", "text", "stars"]

for chunk in pd.read_json(REVIEWS_CLEAN, lines=True, chunksize=chunk_size):
    # keep only users in the selected communities
    chunk = chunk[chunk["user_id"].isin(target_users)].copy()
    if chunk.empty:
        continue

    # add community label
    chunk["community"] = chunk["user_id"].map(user_to_comm)

    # run sentiment in batches
    texts = chunk["text"].tolist()
    preds = sentiment(texts, batch_size=32, truncation=True)

    # append predictions
    chunk["sent_label"] = [p["label"] for p in preds]
    chunk["sent_score"] = [p["score"] for p in preds]
```

```
    all_chunks.append(chunk)

reviews_sent = pd.concat(all_chunks, ignore_index=True)

print(f"Total community reviews with sentiment: {len(reviews_sent)}")


out_path = SENT_DIR / "community_reviews_sentiment.parquet"
reviews_sent.to_parquet(out_path, index=False)
print(f"Saved per-review sentiment to: {out_path}")
```

```
Total community reviews with sentiment: 14389
Saved per-review sentiment to:
data/processed/sentiment/community_reviews_sentiment.parquet
```

[140]:
```
reviews_sent
```

[140]:

	review_id	user_id	business_id	stars	useful	funny	cool	text	date
0	fMGOJq3i_DUUukBg7QU9SA	o6UJMpHcpLJEvmKLrxLS3w	0qu0fNTOsSmuREYVIMPuIQ	4	0	0	1	So it is a good friends birthday today and we …	2016-10-14 03:44:53
1	X7TNHS4htMTWSQeBp8LLbw	SJvCPlJ5X6Jkbc0TggIiCg	MaYb7qMN6BomP1zQGj3Wjg	5	0	0	0	Today after our Ethiopian food fiasco (see Sel…	2011-01-09 05:34:28
2	IrQyuZfiMKRkh3V-0gApbA	rpUOaLv_7QkDeFQSZMZhOg	MHqeqoJEjsTHivmuXOW4iQ	4	4	0	0	Really great place for wine!\n\nCame here with…	2013-12-29 01:22:34
3	34EQgWpvduSSEC_qXPDoyQ	o6UJMpHcpLJEvmKLrxLS3w	a1E2BPYnufFzb6YX2pplJg	3	0	0	0		
4	8izdGftq1oOKxOJtw-UAeg	di2EAA36q-RtOU1eaVb_sg	ZiQiMlvgk19GrEEkN9kyRA	1	0	0	0		
…	…	…	…	…	…	…	…		
14384	GUptFS3Yd-RSe2FuOSTxkA	vTprcHGBn9asRfI349M1Fg	kfvpS3wrmGIwl01chveMHA	4	5	1	3		
14385	43p9WzrU8So7XXE3INsTWw	oyXIV-9GRBBZrFydPCaR2w	GXFMD0Z4jEVZBCsbPf4CTQ	4	6	4	9		
14386	oaFnU9AKbRjCDiJRgro0Dw	3UnbQp8x5VhWK0DM6MPD6A	bfh3U-F-TFtP1cC_ExFriQ	5	0	0	0		
14387	WPBp-8bJ0c6-WEARb2SNiw	cDm5lJjmz7jW2-XEgMx6xw	Nj9XbH55RlOQ8yrK32R60w	3	0	0	0		
14388	3JfTIu6z25ddWi_iIyozZA	djxnI8Ux8ZYQJhiOQkrRhA	9SQlv-ogZ7lvqe0yzHd4wg	5	1	0	1		

```
3      Came down here for some of their tasty fish ta… 2013-12-06 23:06:53
4      I have been to The Blind Tiger on multiple occ… 2016-01-17 15:53:55
…                                                    …                   …
14384  Always looking for a good nail salon.   Every … 2019-04-23 18:54:59
14385  We ordered online for pick up after reading th… 2019-09-03 21:27:09
14386  We came here today while browsing with my baby… 2021-08-03 22:42:54
14387  Parking is pretty limited, there's a good amou… 2021-10-10 13:52:06
14388  I met Heather, the Owner of Lemon, at the Mary… 2009-10-20 18:11:20

      community sent_label  sent_score
0         top_1   positive    0.972468
1         top_3    neutral    0.415680
2         top_3   positive    0.983102
3         top_1    neutral    0.479868
4         top_6   positive    0.565938
…         …         …          …
14384     top_4   positive    0.760308
14385     top_1   positive    0.516439
14386     top_3   positive    0.890122
14387     top_5    neutral    0.481260
14388     top_3   positive    0.945490

[14389 rows x 12 columns]
```

[141]:
```
reviews_sent["yelp_label"] = reviews_sent["stars"].apply(ap.label_yelp_liked)

confussion_matrix_like = pd.crosstab(
    [reviews_sent["community"], reviews_sent["yelp_label"]],
    reviews_sent["sent_label"],
    normalize="index",
)
print(confussion_matrix_like)
```

```
sent_label            negative   neutral  positive
community yelp_label
top_1     hated       0.664223  0.149560  0.186217
          liked       0.027936  0.063373  0.908691
          neutral     0.171875  0.173713  0.654412
top_2     hated       0.734483  0.124138  0.141379
          liked       0.020214  0.055291  0.924495
          neutral     0.244318  0.190341  0.565341
top_3     hated       0.737013  0.149351  0.113636
          liked       0.033514  0.056757  0.909730
          neutral     0.270073  0.145985  0.583942
top_4     hated       0.756757  0.156156  0.087087
          liked       0.023856  0.058672  0.917473
          neutral     0.300395  0.130435  0.569170
top_5     hated       0.681319  0.131868  0.186813
```

```
          liked       0.025086  0.076397  0.898518
          neutral     0.195000  0.215000  0.590000
top_6     hated       0.827586  0.086207  0.086207
          liked       0.009288  0.015480  0.975232
          neutral     0.360000  0.160000  0.480000
```

We observe, that the reviews are really consistent, when we look at the positive and negative sentiment scores in the communities. We also notice, that the neutral reviews are slightly positive skewed.

Table 11. Top 3 association rules (by lift) from each global and community-level rule set for liked and hated baskets. Category abbreviations: Am.N = American (New), Am.T = American (Traditional), B&B = Breakfast & Brunch, C&T = Coffee & Tea, IC/FY = Ice Cream & Frozen Yogurt, Mex = Mexican, Sandw. = Sandwiches, Med. = Mediterranean, F&C = Fish & Chips, Steakhs. = Steakhouses, GF = Gluten-Free.

Scope	Sentiment	Rank	Antecedent	Consequent	Support	Confidence	Lift
Global	Liked	1	Am.N, Am.T, B&B	Mex	0.0207	0.5775	3.1323
Global	Liked	2	Am.N, Am.T, Mex	B&B	0.0207	0.6161	3.1219
Global	Liked	3	Am.N, B&B, Mex	Am.T	0.0207	0.7048	2.8893
Global	Hated	1	Am.N, Burgers	Am.T	0.0122	0.4583	1.8704
Global	Hated	2	Am.N, B&B	Am.T	0.0121	0.4207	1.7170
Global	Hated	3	Am.N, Pizza	Am.T	0.0132	0.4194	1.7115
Top 1	Liked	1	B&B, Salad	Am.N, Coffee & Tea	0.0637	0.9412	5.7618
Top 1	Liked	2	B&B, Italian	Am.T, Coffee & Tea	0.0637	0.6957	5.6325
Top 1	Liked	3	Coffee & Tea, Italian	Am.T, B&B	0.0637	0.7619	5.4639
Top 1	Hated	1	Am.N, Am.T, Burgers, Sandw.	Pizza, Seafood	0.0690	1.0000	14.5000
Top 1	Hated	2	Am.N, Pizza, Sandw., Seafood	Am.T, Burgers	0.0690	1.0000	14.5000
Top 1	Hated	3	Am.N, Pizza, Seafood	Am.T, Burgers, Sandw.	0.0690	1.0000	14.5000
Top 2	Liked	1	Am.N, Cajun, Sandw.	Am.T, B&B, Mex	0.1020	0.7143	6.3636
Top 2	Liked	2	Am.T, Mex	Am.N, Cajun, Sandw.	0.1020	0.9091	6.3636
Top 2	Liked	3	Am.N, Cajun, Sandw.	Am.T, Mex	0.1020	0.7143	6.3636
Top 2	Hated	1	Seafood	Mexican	0.1212	1.0000	3.6667
Top 2	Hated	2	Burgers	Mexican	0.1212	0.8000	2.9333
Top 2	Hated	3	Vegetarian	Am.T	0.1212	0.8000	2.0308
Top 3	Liked	1	Am.N, B&B	Am.T, IC/FY, Mex	0.1042	0.7143	6.2338
Top 3	Liked	2	Am.T, IC/FY, Mex	Am.N, B&B	0.1042	0.9091	6.2338
Top 3	Liked	3	Bakeries, C&T	Am.N, Chinese	0.1042	0.8333	6.1538
Top 3	Hated	1	Am.T, Italian	Pizza	0.1176	1.0000	4.8571
Top 3	Hated	2	Am.N, Am.T	Mex	0.1176	0.8000	3.8857
Top 3	Hated	3	Am.T, Pizza	Italian	0.1176	0.8000	3.8857
Top 4	Liked	1	B&B, Sandwiches	Cafes	0.0097	0.7647	2.9632
Top 4	Liked	2	Cafes	B&B, Sandwiches	0.0097	0.8125	2.9632
Top 4	Liked	3	B&B, Sushi Bars	Am.T	0.0097	0.8667	2.6867
Top 4	Hated	1	Burgers	Sandwiches	0.2083	0.6250	2.1429
Top 4	Hated	2	Sandwiches	Burgers	0.2083	0.7143	2.1429
Top 4	Hated	3	Sandwiches	Mexican	0.2500	0.8571	1.8701
Top 5	Liked	1	Am.T, Other	Am.N, Mex, Seafood	0.0714	1.0000	14.0000
Top 5	Liked	2	Am.N, Mex, Seafood	Am.T, Other	0.0714	1.0000	14.0000
Top 5	Liked	3	Other, Seafood	Am.N, Am.T, Mex	0.0714	1.0000	11.6667
Top 5	Hated	1	Am.N, Am.T, C&T, Delis, GF	Fast Food	0.1111	1.0000	9.0000
Top 5	Hated	2	Am.N, Am.T, Delis, Fast Food	C&T, GF	0.1111	1.0000	9.0000
Top 5	Hated	3	Am.N, Am.T, Fast Food, GF	C&T, Delis	0.1111	1.0000	9.0000
Top 6	Liked	1	Seafood	Sandwiches	0.0698	0.7500	5.3750
Top 6	Liked	2	Japanese	Mexican	0.0930	0.8000	3.8222
Top 6	Liked	3	Pizza	Am.T	0.0930	0.8000	2.8667
Top 6	Hated	1	Am.T, Burgers, Diners, Donuts, Med., Steakhs.	F&C, Gastro	0.1429	1.0000	7.0000
Top 6	Hated	2	Am.T, Burgers, Donuts, Gastro, Med., Steakhs.	Diners, F&C	0.1429	1.0000	7.0000
Top 6	Hated	3	Am.T, Diners, Donuts, Gastro, Med., Steakhs.	Burgers, F&C	0.1429	1.0000	7.0000