Content Recommendation Systems:
A Comparison of User-Based and Item-Based Approaches

Robert Juchnicki

Yale University
Adviser: Dana Angluin
CPSC 490

ABSTRACT

As databases and server warehouse technologies continue to become more powerful and sophisticated, enormous quantities of data are being stored about users of mobile and web services and their interactions with these platforms. These vast stores of user data can be used to construct recommendation systems that can help connect users to other content that may be useful or interesting to them. Collaborative filtering algorithms provide the backbone for the current state of the art in recommendation systems. Collaborative filtering requires users to express some ratings or preferences on a subset of the available content, and computes recommendations using this data. In this paper, I explore and analyze the Yelp Dataset Challenge dataset, which contains data about users, reviews, and businesses and their characteristics. Using my findings about the structure of the data and statistical measures of the data, I construct two classes of collaborative filtering recommendation systems: user-based systems and item-based systems. By dividing the businesses from each user’s reviews into a training set and test set, I train recommendation systems, generate recommendations, and test their accuracy against content users have already expressed an interest in. The structure of the project lends itself well to dividing development between three main components: the database builder, the analysis engine, and recommendation engine. The system was built using Python 2.7 and the scientific computing modules NumPy and Matplotlib. The software is cross-platform and works both on Windows and Unix-based operating systems. After giving an overview of the system, I propose future directions to build on this work.

1. INTRODUCTION

A vast quantity of data is produced each day through interactions between users and mobile and web services. Advancements in database systems and computing server technologies are allowing companies to process, store, and organize more data than ever before. The providers of internet services retain and organize large volumes of data about their users. Moreover, companies catalogue and categorize large amounts of data about the items and services they offer, such as products or ratings and reviews. It is possible to leverage these large, rich datasets to create recommender systems that point users to new content tailored to their interests. In fact, as the amount of data web services store continues to increase, it often becomes necessary to use recommendation systems to point individuals to the useful content in what can otherwise be an overwhelming stream of options and information.
2. BACKGROUND

There are various approaches that have been researched and used to create recommendation systems in myriad domains. The most naïve approach is one in which items are chosen at random from the set of all items. This is not an effective approach because as the number of items increases, the probability of selecting a good recommendation usually decreases. As a result, a random prediction algorithm is often used as a baseline which recommendation algorithms are evaluated against. Another technique for producing recommendations is the use of frequent sequences. In this approach, common sequences of interactions with items are used to recommend content after certain sequences of content accesses are performed by a user. Content-based algorithms for recommender systems treat recommendation generation as a search problem for similar items in the set of possibilities. Each item has certain categories such as author, subject, or keywords, and a search is performed for items with the same entries for these categories. Collaborative filtering (CF) requires users to express some ratings or preferences on a subset of the available content. Variations of CF provide the backbone for the current state of the art in recommendation systems. CF requires the following [5]:

- List of users, \( U = u_1, u_2, \ldots, u_m \)
- List of items, \( I = i_1, i_2, \ldots, i_n \)
- For each user \( u_i \), a list of items \( I_{u_i} \) on which \( u_i \) has expressed his opinion

There are two basic classes of approaches to CF techniques: item-based approaches and user-based approaches. Item-based algorithms construct a similarity matrix row by row by computing similarities between items, without comparing items to themselves. Some approaches only retain the greatest \( k \) similarity measures for each row. The following algorithm provides a basic approach for constructing the similarity matrix:

```plaintext
for i = 1 to n
    for j = 1 to n
        if i ≠ j
            \( M_{i,j} = \text{sim}(v_i, v_j) \)
        else
            \( M_{i,j} = 0 \)
    for j = 1 to n
        if \( M_{i,j} \) is not among the \( k \) largest values in row \( i \) of \( M \), then \( M_{i,j} = 0 \)
```

Algorithm 1 – Deshpande and Karypis’s algorithm for constructing a similarity matrix for item-based CF algorithm [3].

The similarity, or closeness of two items can be defined in terms of a distance function. There are many options for similarity measures. To compare tuples of numerical values we can use Euclidean distance or Manhattan distance:

- \( \text{sim}(i, j) = d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + \cdots + (x_{in} - x_{jn})^2} \)
- \( \text{sim}(i, j) = d(i, j) = |x_{i1} - x_{j1}| + \cdots + |x_{in} - x_{jn}| \)

These distances are not appropriate for categorical data such as color or other attribute data [1]. A few commonly used similarities for categorical data are [1]:

- Cosine distance: \( \text{sim}(\vec{x}, \vec{y}) = \cos \theta = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| |\vec{y}|} \)
• Conditional probability-based similarity: measure similarity based on the conditional probability of enjoying (or rating) an item given that the user has already enjoyed or shown interest in another item [1]. \( sim(i, j) = P(i \mid j) \cdot \alpha \), where \( \alpha \) is a factor chosen depending on the problem being solved.

• Pearson correlation similarity: \( sim(x, y) = corr_{x,y} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}} \)

User-based algorithms operate under the assumption that each user of a service belongs to a group of users that exhibit similar behaviors and preferences. User-based CF recommenders profile users to categorize them into groups, determine which items are preferred or well-liked by users in each group and assign weights to them, and generate recommendations based on the items with greatest weight for a user’s group that have still not been consumed by that user. There are some obstacles in scaling user-based approaches [1]. Cluster generation to determine groups scales linearly with the number of users and cannot be precomputed because the sets of users and items are dynamic [1] and the categories by which users are classified might change over time. Thus, user-based approaches prove impractical for large databases of users and items [1]. Some CF algorithms use subsets of items and subsets of users to implement an approach that combines item-based and user-based techniques.

Metrics to evaluate recommender systems include accuracy, efficiency, and scalability. Accuracy is a measure of how closely the output of a recommender system matches users’ preferences. This can be measured in a few ways, depending on the nature of the items and data. These include seeing if a recommendation of an item results in a desirable interaction with the item, such as a purchase, having users evaluate recommendations, and evaluating recommendations against users’ past ratings. Accuracy metrics and narrow similarity comparisons can be limiting and generate outputs that are not beneficial. Recommender outputs are often judged on an item by item basis rather than by the usefulness of the entire set of items [4]. One method to deal with this obstacle is to compute a measure of the diversity of a list of recommendations, and adjust the list until it surpasses a specified threshold [9]. In some cases the most accurate recommendations are not the best. For example, if a travel agency recommends locations that you have already visited, it might not be beneficial to you if you are seeking novelty. Efficiency is measured by the amount of memory and computation time a recommendation requires [1]. Scalability refers to the ability of the system to continue to provide many requests even as the number of users and the quantity of data increases. A tradeoff between prediction accuracy and scalability is often inevitable as the database the recommender operates on grows. This is usually addressed by utilizing only a portion of all the data available with the most similar characteristics [1].
3. DATA

The Yelp Dataset Challenge datasets for users, businesses, and reviews were analyzed and used to create recommendation systems.

3.1 Structure of Yelp Academic Dataset Data

The Yelp datasets are organized in multi-item JSON files that store one piece of information per line. In this study, the JSON files were converted into CSV files for ease of parsing using a JSON to CSV conversion utility released by Yelp on GitHub [8]. The datasets contain entries for 366715 users, 61184 businesses, and approximately 1.6 million reviews, each labelled by their type and a unique id among entries of their type.

User entries store the month and year that a user joined Yelp, their first name, a review count, and an average star rating for the user’s reviews. In addition, users have a list of years that they were Elite users, as awarded by Yelp. Yelp awards Elite status to users that it deems to have authenticity and an unbiased approach in producing content and interacting with businesses, a large volume of activity on the service, and a strong connection to the Yelp community [6]. Users can also nominate other users for Elite status. Lists of ids for friends that users have connected to and fans they have made on the service are also stored, providing information about the users’ social graph.

Business entries contain detailed location information including the businesses’ city, state, full street address, and latitude and longitude coordinates. Additionally, the neighborhoods in which the businesses have branches are stored in a list if there is more than one branch. Businesses are labelled by their name in addition to their unique ids. Each business has a review count and an overall star rating based on reviews. Opening and closing times are also listed in the entries. The dataset contains detailed information about business attributes stored in Boolean entries. These Boolean fields relate to ambience, specialties, catering to dietary restrictions, payment types accepted, and other characteristics. Each business also has a list of categories that describe it, containing entries from 783 possible options.

Reviews entries contain ids for the user and business that they are related to. All reviews are labelled with a review id and dated. Each review has a star rating, which is a whole number from 1 to 5, and the full text for each review is stored in a text field. Each review also has counts for how many users voted for it as “cool”, “useful”, or “funny”.

The datasets do not offer any information about user interaction with the Yelp mobile or web applications that can be used to generate implicit information for training recommendation systems. The data gives no window into users’ interactions with businesses and ratings on Yelp’s platform beyond the ratings, reviews, and check-ins they have made. A complete list of data fields stored for each type of entry can be found in db_builder.py in the source code under the names USER_FIELDS, BUSINESS_FIELDS, and REVIEW_FIELDS.

3.2 Initial Analysis

Initial tallying and statistics gathering was critical in beginning to understand important characteristics of the data that would guide the architecture of the recommendation engine. The mean review count for users is 32.2, while the 25-th percentile review count is 2.0, the median review count is 6.0, and the 75-th percentile review count is 21.0. This huge gap between the mean
and the median, and even the mean and the 75-th percentile mark, indicates that a relatively small portion of the users probably produce a substantial fraction of reviews on Yelp. In fact, the top 4.57% of users (by review count) produce 50% of reviews. A histogram of review counts for users for counts between 0 and 100 is shown below:

![Histogram charting the frequency of review count values for users in the Yelp Dataset Challenge data.](image)

Figure 1 – Histogram charting the frequency of review count values for users in the Yelp Dataset Challenge data.

It is likely that most users are only reviewing some of the businesses whose services they have used and are using Yelp primarily to view information and content other users have produced. This is a feature that could create some difficulty in producing accurate recommendations for many users in the absence of a source to collect implicit data from such as business profiles that users have viewed.
Businesses follow a similar trend for review counts as shown in the histogram below:

![Histogram charting the frequency of review count values for businesses in the Yelp Dataset Challenge data.](image)

The mean review count for businesses is 28.7, while the 25-th percentile review count is 4.0, the median review count is 8.0, and the 75-th percentile review count is 21.0. A similar skewing of the data as that discussed for user review counts is observed; however, it is not as extreme for businesses.
The star ratings for reviews are found to be skewed high as seen in the histogram below:

**Frequency of Star Ratings in Reviews**

The mean star rating for reviews is 3.74. The 25-th percentile rating is 3.0, the median rating is 4.0, and the 75-th percentile rating is 5.0. The most likely explanation for this is that businesses that do not receive high ratings will either improve aspects of their business or go out of businesses.

Another consideration that needed to be addressed in the software, as highlighted by initial analysis, was users who reviewed businesses in multiple cities and states. There were 54474 users (14.9%) who reviewed businesses in exactly two cities and 31643 users (8.6%) who reviewed businesses in more than two cities. There were 17619 users (4.8%) who reviewed businesses in exactly two cities and 1732 users (0.5%) who reviewed businesses in more than two states. Recommended businesses should definitely reside in the state that the user is determined to currently reside in to be most relevant. Recommendations can be made in any city in the user’s current state and will naturally be weighted toward the cities in which the user most commonly reviews businesses since a user’s reviews will be divided into a training and test set randomly.

Data about Elite users can allow for the potential to weigh the input of some users that have been recognized as positive presences in the community more than others. Approximately 25,000 users (6.9%) were Elite users at some point, while approximately 11,000 users (3.1%) were Elite as of the compilation of the datasets in 2015. Elite status was awarded from 2005 to 2015, with 2004 as the only year in the datasets without any elite users.

If one observes the results of the analysis, she will find a discrepancy between the review counts obtained by looking at data in each dataset individually. There are 1,569,264 reviews in the review dataset. Meanwhile, the users in the user dataset have produced 11,813,654 reviews and the businesses in the business dataset have received a total of 1,729,825 reviews. This likely results
from the way in which Yelp took cuts of the full data to compile these public datasets. It appears as though the preparers of the datasets first chose a subset of businesses, removed some of the reviews based on some characteristics, and finally compiled the dataset of users from the remaining businesses.

Additional results and more details from inspecting the data and computing statistical measures on it can be found in Appendices A-E.

4. SYSTEM DESIGN & IMPLEMENTATION

The system was primarily built using Python 2.7 and the scientific computing modules NumPy and Matplotlib. The software is cross-platform and works both on Windows and Unix-based operating systems. The system can be broken down into three major components that divided the development into three primary phases: the database builder, the analysis engine, and recommendation engine.

4.1 Database Builder

As described earlier, the datasets were initially stored in multi-item JSON files, and were converted to CSV files using code from the Yelp Github page [8]. Once the files were converted to CSV format, they were parsed as a stream using the built-in Python csv module’s reader functionality. Three databases were created: one for users, one for businesses, and one for reviews. The databases were constructed as key-value databases using Python dictionaries. The keys were each dataset’s entry ids and the values were dictionaries containing the data for the corresponding entries. The general organization of the databases is shown below:

```python
# The user database is created as a dictionary in the following form:
#
# users = {
#     user_id: {
#         'yelping_since': 'YYYY-MM',
#         'review_count': ..., 'average_stars': ..., 'elite': [year1, year2, ...], 'name': ...,
#     }
# users_id: [...]
# ...}
```

Although the CSV files were read as streams and some information was discarded while constructing databases, Python MemoryErrors were encountered due to the large size of the datasets being stored in the dictionary databases. The primary cause of the MemoryErrors was the large size of the strings storing reviews’ texts; however, when a few new necessary entries were added to some of the databases in later phases, MemoryErrors arose again. The text field was discarded for local analysis, and the code and datasets were moved from a 4GB RAM machine to a 32 GB RAM machine for the capability of working with all of the databases at the same time. The data values in the original datasets were initially stored as strings, and were converted to their original data types using the built-in ast (Abstract Syntax Tree) module. Once the datasets were properly parsed and the databases were compiled, they were saved using the Python module cPickle. cPickle is a C implementation of the module pickle, which implements an algorithm for serializing Python objects. This module stores objects’ hierarchies as byte streams and converts byte streams back into object hierarchies.
4.2 Analysis Engine

The analysis engine used the databases compiled and saved by the database builder by loading them with the cPickle module. The analysis code used built-in Python functionality and the NumPy module to implement subroutines for database inspection and analysis. The functionality included subroutines to:

- compute the average for a given numerical field for all entries in a database
- compute percentiles values for a given numerical field for all entries in a database
- compute the minimum and maximum of a given numerical field for all entries in a database
- return a list of all unique values stored for a given field for all entries in a database
- return a list of all unique years of dates stored in a given field for all entries in a database

Analyses combining inspection of multiple databases were primarily conducted to determine information about the cities and states that users reviewed businesses in.

The analysis system also included code to produce plots from the datasets. Numpy and the plotting module Matplotlib were used to produce diagrams that assisted in analyzing the data. Histogram data and appropriate bins were computed using NumPy’s histogram functionality. Appropriate bar widths were computed based on the bin information returned for the histogram. These values were then passed to the bar graph plotting function in Matplotlib. Appropriate adjustments were made to the axes’ scales and labels and the figure’s shape using Matplotlib’s graphing functionality.

The reader can refer to figures in Section 3 and Appendices A-E to see more information about the specific results that were computed by the analysis engine.

4.3 Recommendation Engine

The recommendation engine begins by initializing the recommenders. The initialization begins by computing fields for each user and saving them in the database to improve the efficiency of recommender creation. These are the user’s current state and a list of review ids for reviews written by the user. It is important to determine the user’s current state to compute relevant recommendations. The determine_state method takes a user_id and returns the predicted current state. The current state is computed by sorting all of a user’s reviews by date and returning the state that appears the greatest number of times in the most recent half of all these reviews.

A sample of the determine_state function can be found in Appendix F. This code highlights the workflow of accessing and interacting with multiple databases in the system and the ease of prototyping in Python afforded by an ease of converting and moving between data types and object types.

The recommender initializer has functionality for basic vector operations: computing the norm, normalizing a vector, taking the dot product of two vectors, and computing the cosine distance between two vectors. A helper function, attribute_vector, builds a vector of 0’s and 1’s, denoting the presence or absence of attributes and categories passed in as arguments, to describe a business. compute_similarities constructs a similarity matrix for all businesses in a given state by performing similarity calculations using a given similarity measure, function for making vectors for businesses, attributes, and categories and returns it along with a list of business_ids to label the matrix by both indices and business_ids. Similarity matrices and their business labels are saved using cPickle.
The recommend script computes recommendations using the recommenders trained in the recommender initialization. The recommend script also implements a randomized recommendation algorithm that returns a random subset of size k of all businesses in a state to use as a baseline in recommendation trials. The script runs trials by state, creating recommendations for all users currently in the state by using an item-based recommender and the random subset algorithm. For item-based recommendations, it first sorts all businesses the user being considered has reviewed in the given state by the user’s ratings. Next, for each of the top three rated businesses, it determines the three most similar businesses based on a similarity matrix. The script then checks to see if any of the recommendations produced by either approach match any of the businesses the user has reviewed. If there is a match, it is recorded. After the trial an accuracy percentage is outputted for each method.

5. RESULTS

An item-based algorithm and a random subset algorithm were run for users and businesses in Illinois, Wisconsin, Pennsylvania, and North Carolina. Businesses were characterized by ambience, “good for”, and dietary needs served attributes, as well as a limited set of the 783 available categories: Drugstores, Grocery, Buffets, Restaurants, Food Stands, Bistros, and Hotels & Travel. The similarity between businesses was calculated using a cosine distance. Nine random subset recommendation trials were run and averaged. The following initial results were obtained:

<table>
<thead>
<tr>
<th>State</th>
<th>Random Algorithm Accuracy</th>
<th>Item-Based Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>3.56%</td>
<td>6.53%</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>1.29%</td>
<td>4.46%</td>
</tr>
<tr>
<td>Pennsylvania</td>
<td>1.10%</td>
<td>2.84%</td>
</tr>
<tr>
<td>North Carolina</td>
<td>0.61%</td>
<td>2.11%</td>
</tr>
</tbody>
</table>

Table 1 – Table of Item-Based experiment results compared to a randomized algorithm baseline.

A second item-based trial was run for Illinois using the same attributes as in the first trial and all available categories. An accuracy of 9.12% was achieved.

A third item-based trial for Illinois made use of all available attributes and categories, achieving, an accuracy of 7.31%.

6. DISCUSSION

The limited set of attributes and characteristics was chosen in the first trial because it is computationally expensive to run many vector calculations on extremely high-dimensional vectors. This choice allowed for trials to be run more quickly for initial experimentation and fine-tuning of the recommendation engine. Illinois was used for further trials testing the use of different attributes and categories because it had the least businesses of the states used, and thus, its similarity matrix was the fastest to construct. The attributes used in the first trial and second trial were selected because they were the ones that would seem logically to offer the most nuanced information to differentiate between businesses. The categories used in the first trial were chosen
because they were the broadest categories available that would potentially be useful in differentiating between businesses. Categories outside these broad ones are only useful if all relevant niche categories are present. For example, it is not helpful to include the category for Mexican food unless you include all categories describing the region or ethnicity of food served at a restaurant or to include dermatologists unless you include all categories describing medical practices. When all categories were used for Illinois, a 39.66% increase in accuracy was experienced. When all attributes and categories were used, only a 12.94% increase in accuracy was experienced compared to the initial trial. This decreased performance compared with a more selective use of attributes demonstrates that using more information does not necessarily improve the ability to differentiate between items to produce better recommendations.

7. FUTURE WORK

Due to time constraints, user-based recommenders were not implemented. They will be implemented in the future and trials will be run to compare their performance to the item-based recommenders that were implemented. Furthermore, experimentation with different similarity measures remains to be performed.

Due to the rich nature of the Yelp Dataset Challenge data, a variety of directions for future work and improvements to the recommendation system implementations exist. One of these is the incorporation of data from textual analysis of review texts. Textual analysis can be performed to compute additional measures of how useful a given review might be for other users. Such a text parsing system could be trained by utilizing user feedback in the form of “useful” votes on reviews, which are listed in the datasets. Additionally, sentiment analysis of review texts could be used to compile a second star rating for any given review. Another potential direction is creating a fraud detection system, which incorporates textual analysis to remove fraudulent or biased reviews.

Another promising area for future work is the use of social graph data to attempt to improve recommendation creation. Analyses of the social graphs contained in the datasets can possibly reveal some users as figures of authority in the Yelp community. Such analyses can also be used to give greater weight to the businesses one’s friends and individuals one is a fan of have reviewed, which may give rise to better recommendations under certain conditions.

Aside from possibilities to leverage additional fields in the datasets and analyses of the datasets, there are a few key opportunities to improve the technical aspects of the software. Python was a useful tool for rapidly prototyping throughout all phases of development, especially given the time constraints of an academic semester for the project. The wealth of existing scientific computing and mathematical tools available in Python modules was extremely valuable too. However, since Python is a scripting language, it suffers from speed drawbacks, which can become very restrictive when dealing with large datasets. Some brief experimentation with Cython was performed to try to improve the run times of the scripts. Cython is programming language that aims to be a superset of the Python language and allows for static type declarations of C datatypes in Python code and static compilation of Python code. Cython source code is translated into optimized C/C++ code and is compiled into Python extension modules [2]. These features can result in faster program execution. Unfortunately, there was not enough time to port the whole codebase to Cython. It may also be more valuable to make use of a different compiled language with faster execution speeds and other desirable features and libraries for future iterations of the code. Two of the most expensive actions in the current code are cPickle loads and dumps. In the future, it would be good to move the datasets into a formal database system.
8. ACKNOWLEDGEMENTS

I would like to take a moment to thank Professor Dana Angluin for her guidance as my advisor throughout the semester of work on this project. Our frequent meetings helped keep this project on track and sparked valuable insights that moved it forward. I also want to thank Winnie Wang for reading through the final draft of my project report. Finally, I would like to thank Winnie and my other close friends for lending me their ear from time to time and offering their support throughout this process.

9. REFERENCES


APPENDIX A: USER STATISTICS

Number of users in database: 366715

**The top 4.57% of users (by review count) produce 50% of reviews!**

Mean Review Count: 32.21
25-th Percentile Review Count: 2.0
50-th Percentile Review Count: 6.0
75-th Percentile Review Count: 21.0
80-th Percentile Review Count: 29.0
81-st Percentile Review Count: 32.0
85-th Percentile Review Count: 44.0
90-th Percentile Review Count: 74.0
95-th Percentile Review Count: 151.0
Min Review Count: 0
Max Review Count: 8843

Mean Average Star Rating: 3.72
25-th Percentile Average Stars: 3.27
50-th Percentile Average Stars: 3.86
75-th Percentile Average Stars: 4.43

Number of users that were elite at some point: 25301
Number of users who had elite status in the beginning of 2015: 11386
All years any users were elite: 2005-2015

Statistics for year joining Yelp
2004: 51 users
2005: 691 users
2006: 3974 users
2007: 10676 users
2008: 19390 users
2009: 32968 users
2010: 50722 users
2011: 69210 users
2012: 63897 users
2013: 63483 users
2014: 50505 user
APPENDIX B: BUSINESS STATISTICS

Number of businesses in database: 61184
Number of businesses that are open: 53725
Number of businesses that are closed: 7459

Mean Review Count: 28.27
25-th Percentile Review Count: 4.0
50-th Percentile Review Count: 8.0
75-th Percentile Review Count: 21.0
Min Review Count: 3
Max Review Count: 4578

Mean Average Star Rating: 3.67
25-th Percentile Average Stars: 3.0
50-th Percentile Average Stars: 3.5
75-th Percentile Average Stars: 4.5

Number of states businesses are in: 26
Number of cities businesses are in: 378

Number of categories for businesses: 783
APPENDIX C: REVIEW STATISTICS

Number of reviews in database: 1569264

Mean Stars: 3.74
25-th Percentile Stars: 3.0
50-th Percentile Stars: 4.0
75-th Percentile Stars: 5.0

APPENDIX D: CROSS-ANALYSIS AND EXPLORATION OF DATASETS

Users with reviews in two cities 54474
Users with reviews in more than two cities: 31643

Users with reviews in two states: 17619
Users with reviews in more than two states: 1732

APPENDIX E: NUMBER OF BUSINESSES IN EACH STATE USED

IL: 627
WI: 2307
PA: 3041
NC: 4963
APPENDIX F: CODE SAMPLE FROM initialize_recommenders.py

```python
# Return the key in a dictionary with the max value.
def key_for_max_value(d):
    values = list(d.values())
    keys = list(d.keys())

    return keys[values.index(max(values))]

# Returns the user's current state. Looks at the dates and locations of all
# business the user has reviewed to compute the result.
def determine_state(user_id, users, reviews, businesses):
    review_info = []

    for review in users[user_id]['reviews']:
        business_id = review[review][business_id]
        date_string = reviews[review][date]
        day = date(int(date_string[0:4]), int(date_string[5:7]), int(date_string[8:]))

        review_info.append((review, day, businesses[business_id][state]))

    review_info.sort(key=lambda x: x[1])  # sort review_info tuples by date

    # determine the most common state among the most recent half of reviews
    n = len(review_info)
    states = {}
    for i in range(n//2, n):
        state = review_info[i][2]
        if state in states:
            states[state] += 1
        else:
            states[state] = 1

    return key_for_max_value(states)
```