**Description of the problem**

Customer profile information is highly coveted by brands, advertisers, marketers, match-makers, etc. in order to improve targeting, conversion, ROI, etc.; however, such data can be expensive and stale, assuming you even have the budget to pay for it.

With the recent spate of bad publicity (and deservedly so), Facebook and other social media/network platforms have significantly limited the breadth and depth of the information they make available to third party developers and researchers via their APIs. For companies whose business models depended on these APIs, they are scrambling to fill this massive gap or have already gone out of business.

For limited applications, i.e. not hundreds of millions of users, web-scraping can still allow companies to compile richly textured customer profiles, from which they can build their own network graphs, predictive analytics, etc.

**Why the problem is interesting**

The better a company knows its customer, the better it is able to personalize its product and marketing to that customer. Especially in a world full of attention deficit, being able to say/do/show the right thing to the right person at the right time has never been more critical. And while I’m not advocating a world in which society is even more inundated with marketing and advertisements, the customer experience does benefit from smart, context-specific information, e.g. if I happen to stop by CSV to pick up some cold medicine, and my smartphone wallet automatically applies a coupon to my purchase as I’m checking out, I would really appreciate that.

**What other approaches have been tried**

As mentioned above, social media/network platforms, e.g. Facebook, Instagram, Twitter, etc., provide APIs for third party developers and researchers to obtain information from the platforms directly. This information used to include information about users and other users connected to a particular user along with information about trending topics and which users were following them. With the increased scrutiny on data privacy, these platforms have significantly limited the type of data available via their APIs. Generally speaking, information about a user’s network, i.e. other friends, followers, etc., is no longer available. Even specific information about a given user, e.g. interests, topics followed, etc., often now require user permission. These changes are all great for data privacy but make it harder for third parties to develop rich personas of these people – which is of course the point of data privacy.

Facebook does offer some demographic data to apps and sites that use Facebook login, but that data has also become more limited and can often be quite stale. Surveys on social media, IRL (in-real-life) focus groups, in-product questionnaires all provide some value, but are limited by selection bias, scalability and potential UX issues, respectively. That said, any sophisticated product and/or marketing effort ought to include a combination of these various approaches to ensure all blind spots are covered.

**Hypothesis on why the proposed solution will improve or solve the problem**

As my research into customer personas evolves, I expect to flesh out the customer persona framework below:

* Demographic information: this includes basic information about where a person is front, what school(s) s/he went to, race & ethnicity, etc.
* Hobbies & interests: this is rather self-explanatory but adding the dimension of time makes it more complex, i.e. separating current hobbies & interests from those of the past
* Event-/time-based attributes: this includes events like graduating from college, having a child, etc.; whereas knowing someone is a parent is interesting, know that they just had a child entails a much more specific context
* Evolution of social networks: similar to the event-/time-based attributes, social networks evolve over time; as the saying “birds of a feather flock together” suggests, information about a person’s social network (especially the most recent information) can shed a lot of light on his/her preferences

Realistically given the timeframe of this research project, I expect to be able to generate profiles for demographic information and hobbies & interests. Event-/time-based attributes and social network evolution require persona data to be captured over time and appended to when new information is available. This would require more sophisticated information capture, timestamping, storage and analytical capabilities.

In terms of approach and tools, I intend to predominantly use Python – Selenium for web scraping, Networkx for network graphs, Scikit for any potential modeling, image recognition and prediction. The order of operations will proceed as follows:

1. Build web scraper to scrape social media/network sites for user data
2. Build network graph based on user network (users as nodes and connections as edges) and enrich network with user-specific attributes (node-level attributes)
3. Develop potential features from network graph data for modeling purposes
4. Apply modeling techniques to flesh out personas for users within network graph
5. Share thoughts on how effective these techniques were in creating personas rich enough for commercial purposes

Aside from the obvious limitation of what information is publicly accessible on user profiles on these social media/network platforms, other potential limitations include insufficient storage and processing capacity to really give this a true shot. AWS or Google Cloud may address this if it’s not cost prohibitive for the purposes of a research project.

**1 Build web scraper to scrape social media/network sites for user data**

As the largest social network on the planet, Facebook is as good a place to start gathering user data as another. Facebook users of course have ‘friends’ on the platform and also have access to various information about each friend. For example, for each friend on the platform, we are able to see that friend’s friends. See the screenshot below for an example.

A screenshot of a cell phone

Description automatically generated

In addition to seeing friends of friends, we are able to see some basic information about each of these primary and secondary friends within the “About” section of each user’s profile page. See the screenshot below for an example. As you can see, the “About” section includes information like current/previous occupation, school and major of study, location of residence, relationship status, birthdate and also links to other social media/network pages, e.g. Instagram, LinkedIn, etc. Even with this information, I was able to assemble relatively rich data for our user personas.

A screenshot of a cell phone

Description automatically generated

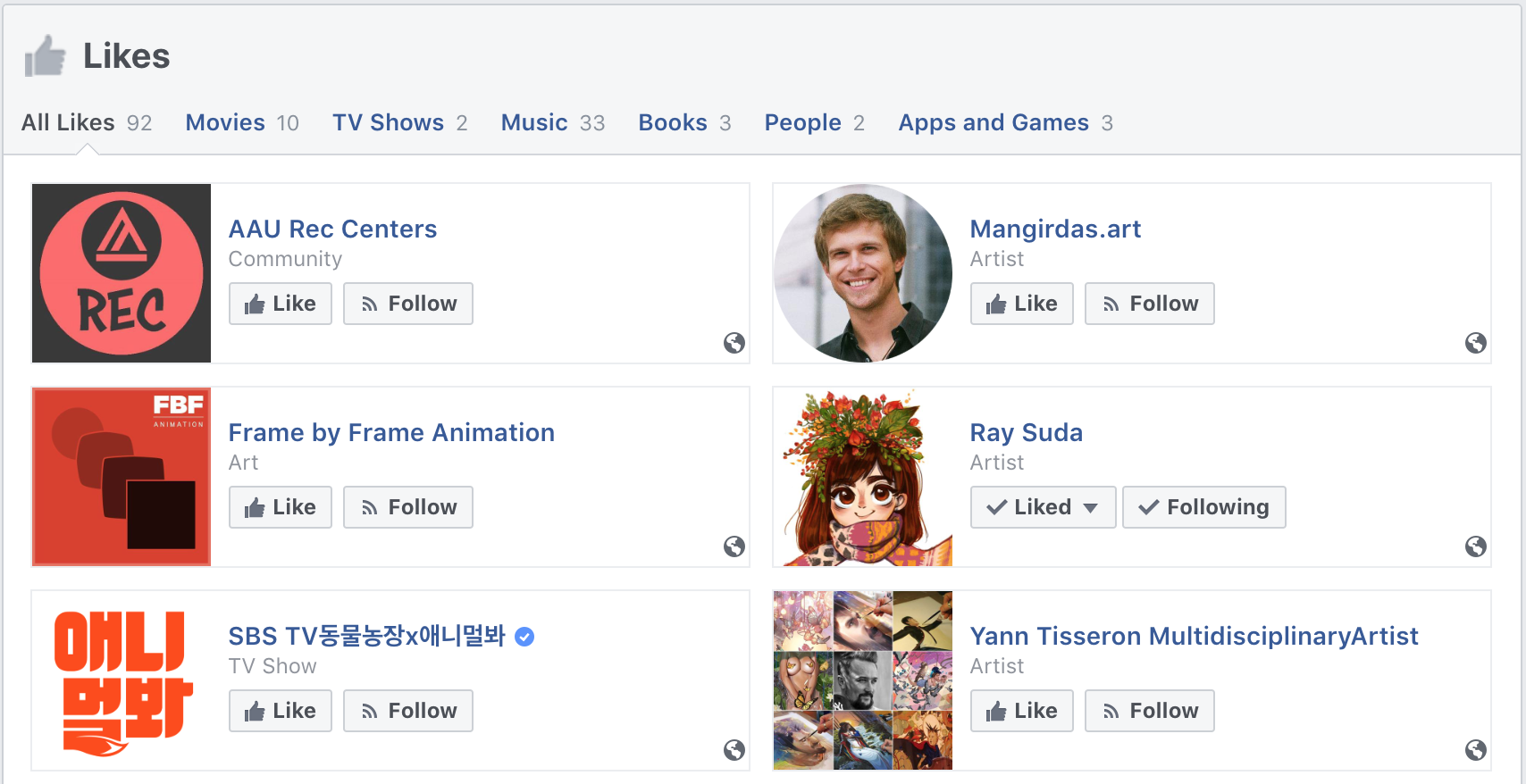
In order to capture this information, we fortunately have Selenium. Selenium is a browser automation tool with an easy-to-use library for Python. It allows us to easily and programmatically navigate websites and access their underlying components, e.g. HTML tags and objects like forms, buttons, etc. The code snippet below includes a function that takes as inputs a Networkx graph object (Networkx is another Python library that allows the creation of network objects) and center (the user from which we will build our network) as arguments. Some of the lines are commented out for expediency; for example, upon first loading a user’s friends page, only the first 20 or so friends are displayed, but the commented-out code will expand the list of displayed friends fully in order to capture all friends of our user. This obviously takes much more time, especially when users have thousands of friends. Nevertheless, the point is that with a snippet of code we are able to quickly start assembling a social network around a given user, and this can ‘snowball’ (which is a specific method to build out a network starting from a central point) into a very broad and deep network – really as broad and deep as the developer/research has time and resources to devote to the cause.

|  |
| --- |
| # define function to iterate through friends list  def get\_friends(g, center):  driver.get('https://www.facebook.com/' + center + '/friends')  time.sleep(random.choice(range(100, 300)) / 100)  friends\_df = pd.DataFrame({'user\_name': [], 'user\_id': []})  # load more friends until all are loaded  # try:  # friends\_n = driver.find\_element\_by\_class\_name('\_3d0').text  # refresh\_n = int(np.ceil(int(friends\_n)/20))  # except:  # refresh\_n = 20 # default to 20 if number of friends is private  # for i in range(refresh\_n):  # actions.send\_keys(Keys.END)  # actions.perform()  # time.sleep(random.choice(range(120, 180)) / 100)  full\_friends\_list = driver.find\_element\_by\_class\_name('\_5h60')  sub\_friends\_list = full\_friends\_list.find\_elements\_by\_class\_name('uiList')  row = 0  for i in range(len(sub\_friends\_list)):  friend\_container\_list = sub\_friends\_list[i].find\_elements\_by\_class\_name('\_698')  for j in range(len(friend\_container\_list)):  friend\_container = sub\_friends\_list[i].find\_elements\_by\_class\_name('\_698')[j]  friend\_user\_name = friend\_container.find\_elements\_by\_class\_name('fsl')[0].text  friend\_href = friend\_container.find\_elements\_by\_class\_name('\_5q6s')[0].get\_attribute('href')  if friend\_href is not None:  friend\_user\_id = re.search("https://www.facebook.com/(.+?)\?", friend\_href).group(1)  g.add\_edge(center, friend\_user\_id) # add the edge to the network  friends\_df.loc[row, ['user\_name', 'user\_id']] = [friend\_user\_name, friend\_user\_id]  print('%d: %s' % (row, friend\_user\_id))  row += 1  # get attributes for center node  # info\_dict = get\_attr(center)  # for key, value in info\_dict.items():  # g.node[center][key] = value |

Similarly, another snippet of code allows us to capture information on the “About” page of a given user. I use this function to capture the occupational, academic and other demographic information about our users. This snippet can be expanded to include whatever information is available on a given social media/network site, but even with this bare-boned approach, I was able to gather some very interesting information.

|  |
| --- |
| # define function to get attributes of friends  def get\_attr(center):  driver.get('https://www.facebook.com/' + center + '/about')  time.sleep(random.choice(range(300, 700)) / 100)  info\_dict = {}  # get contact info and birthday if available  for i in range(1, 5):  try:  info = driver.find\_element\_by\_css\_selector('.\_2pif:nth-child(' + str(i) + ') .\_ikh , .\_2pif:nth-child(5) .\_2ieq div').text  info = info.split('\n')  info\_dict[info[0]] = info[1]  except:  pass  # get workplace, school and home if available  for i in range(0, 8):  try:  info = driver.find\_elements\_by\_css\_selector('.\_5y02 , .\_2pi4')[i].text  if info[0:4] == 'Work':  info\_dict['Work'] = info  if info[0:4] == 'Stud':  info\_dict['Study'] = info  if info[0:4] == 'Live':  info\_dict['Live'] = info  except:  pass  print(center, info\_dict)  return info\_dict |

Midway through this project, I realized that while the information above is certainly interesting, it’s commercial use may be somewhat limited. Therefore, I went back to the publicly available Facebook data to see what other information might be more commercially useful. Fortunately, one of Facebook’s most successful innovations is the Like button. On each user’s profile, Facebook summarizes a user’s history of Likes and groups them into various categories, e.g. movies, TV shows, music, etc., as shown in the screenshot below. Furthermore, when drilling down into a particular category, the specific movies, TV shows, songs, etc. are shown as well, allowing researchers to understand not just that a user likes music, but what genres, artists and songs. With the inclusion of these data, we are able to develop much richer (and commercially useful) personas for the users we are looking at.



Similar to the network and demographic data, we were able to develop a script to scrape these Like data. For purposes of this research project, we limited the data to just the categories and their respective counts, but if we were developing this for commercial purposes, we would certainly have captured the detailed Like data as well.

def get\_likes(center):

driver.get('https://www.facebook.com/' + center + '/likes')

time.sleep(random.choice(range(300, 700)) / 100)

info\_dict = {}

try:

likes = driver.find\_element\_by\_xpath('//\*[@id="pagelet\_timeline\_medley\_likes"]/div[1]/div[2]/div[1]')

likes\_list = likes.find\_elements\_by\_class\_name('\_3c\_')

for e in likes\_list[1:]: # skip all likes

info = re.split("(\d+)", e.text)

info\_dict['Like: ' + info[0]] = info[1]

except:

pass

print(center, info\_dict)

return info\_dict

Of course, the challenge with web-scraping is that scripts can easily break whenever websites changed in any structural way. Usually, only minimal changes are needed to get them working again, but it is a pain and something one has to stay on top of.

**2 Build network graph based on user network (users as nodes and connections as edges) and enrich network with user-specific attributes (node-level attributes)**

With the code snippets above, I gathered a number of potentially interesting attributes (not all attributes were available for all users), such as:

* Current / past places of residence
* Current / past places of work, including companies and job titles
* Current / past places of study
* Area(s) of study, i.e. majors and minors
* Current relationship status and partner information
* Birthdate
* Like data, e.g. number Likes for various categories

Often for network analysis, it is easier to convert individual attributes as different node types. For illustration and simplicity purposes, I’ve only converted the place of study attribute into a separate node type in the below chart, which now includes the user as one node type (the red squares) and the place of study as the other node type (the blue circles).

The structure of the network graph below may at first appear abnormal with its central school node, the inner ring of user nodes, and the outer ring of school nodes. However, given the way the data was assembled, i.e. starting with one users and using the snowball method to build the network around this initial user, it makes sense that many of the connected users all went to the same school (the central node). In this case, the school is the Academy of Art University (AAU) in San Francisco. Interestingly, we see that there is relatively little overlap in the other schools that these users attended. Again, this makes sense since AAU actually attracts students from all over the world in contrast to many state and local schools. As such, these users went to high school and other universities all over the world and really only have AAU in common as a place of study. Conversely, if we looked at a state school in a smaller community, like the College of San Mateo, a community college located in San Mateo, CA, we’d likely find that many of the students there went to the same high schools and came from the same or similar communities.

A picture containing screenshot

Description automatically generated

After considering various approaches to developing rich user personas, I decided to use a recommendation algorithm based on the attributes and Like data that I collected and aggregated as my main approach. Specifically, I used User-Based Collaborative Filtering (UBCF), a type of recommendation algorithm. I’ll go more into the specifics of UBCF in the next section, but here I summarize some of the attribute and Like data that the recommendation engine will ultimately consume.

After snowballing, pruning and attribute collection, the final data set includes 82 users with 144 unique attributes. The table below summarizes the number of users reporting each attribute and the variety of attributes reported. Of the 82 users in the data set, 38 reported their year of birth and 74 reported their places of residence and study. Similarly, there were 18 unique years of birth, 18 unique places of residence and 98 unique places of study. For the Like data, 56 users reported at least one Like category, and there were 9 total categories of Likes, i.e. Movies, Music, Books, TV Shows, Sports Teams, Apps and Games, Athletes, Restaurants and People.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Total Users** | **Users Reporting Attribute** | **Variety of Attribute Values** |
| Year of Birth | 82 | 38 | 18 |
| Place of Residence | 82 | 74 | 18 |
| Place of Study | 82 | 74 | 98 |
| Like Categories | 82 | 56 | 9 |

**3 Develop potential features from network graph data for modeling purposes**

As mentioned above, I decided to use user-based collaborative filtering to predict user preferences across of the various Like categories available on Facebook. For example, some users actively use the Like button, and thus we have some data on these users’ preferences. Many other users do not actively use the Like button, so we know less explicitly about their preferences. However, we have much of the same friend-to-friend network and demographic data for almost all users. I intend to use these common data to determine which users are most similar to one another, and then use that to derive the missing preference data.

In order to do this, I had to develop features from the friend-to-friend and demographic data. Much of this feature engineering was less engineering and more data clean-up and standardization. For example, a user’s place of study can be reported in a variety of ways, e.g. “Studied at Academy of Art”, “Studied at Academy of Art From Feb 2016”, “Class of…”, etc. For each of the attributes collected, a similar regular expression based script was used to isolate just the educational institution’s name.

schools\_list\_clean = []

for study in schools\_list:

# most recent school

study = re.sub(r"Jan |Feb |Mar |Apr |May |Jun |Jul |Aug |Sep |Oct |Nov |Dec ", "", study)

if len(re.findall("(Past:|Class of|\d{4} to)", study)) == 0:

school = re.findall(" at (.\*)", study)

schools\_list\_clean.append(school[0])

else:

school = re.findall(" at (.\*)\s(Past:|Class of|\d{4} to)", study)

schools\_list\_clean.append(school[0][0])

# previously attended schools

if len(re.findall(" Past: (.\*)", study)) > 0:

study\_and = re.findall(" Past: (.\*)", study)

if len(re.findall("(.\*) and (.\*)", study\_and[0])) > 0:

schools\_list\_clean.append(re.findall("(.\*) and (.\*)", study\_and[0])[0][0])

schools\_list\_clean.append(re.findall("(.\*) and (.\*)", study\_and[0])[0][1])

else:

schools\_list\_clean.append(study\_and[0])

schools\_list\_clean = list(set(schools\_list\_clean))

return(schools\_list\_clean)

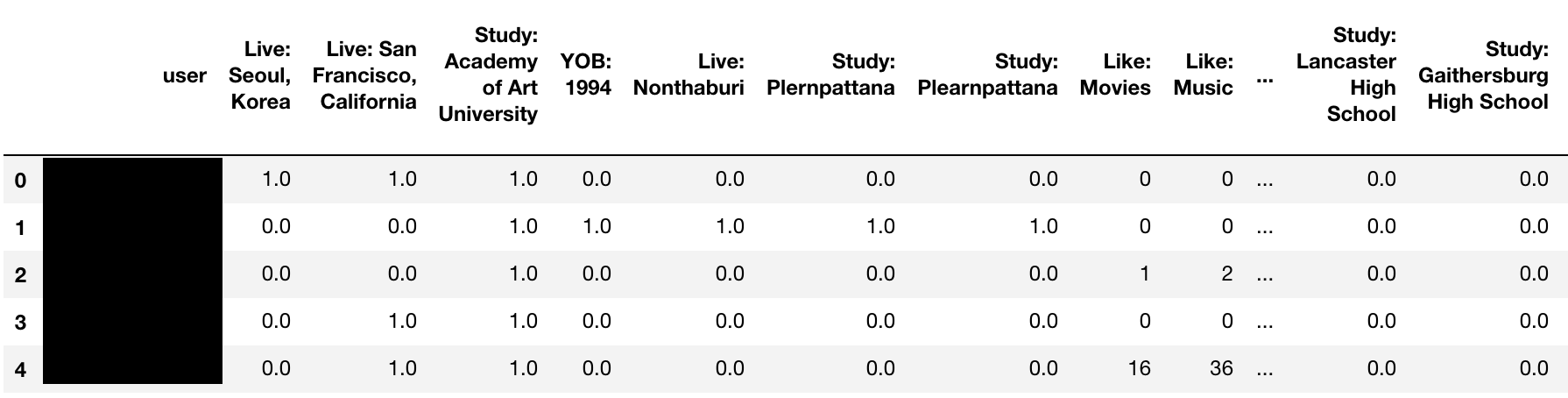
Like data is reported as the specific thing being Liked. For example, a user may have Liked the books “The Hunger Games” and “The Hobbit”. In some cases, users may have Liked hundreds of different books and in some cases no books at all. While the specific titles of books are important in their own right, my focus here is more on the categories of Likes, e.g. Books vs. Movies. As such, I am approximating a user’s interest in a given category based on the number of things Liked within a given category. So specifically for each user, I aggregated the number of Likes within each category. I then normalized all attributes including Like aggregations to fit between 0 and 1, so that no one attribute would overwhelm the others.

**4 Apply modeling techniques to flesh out personas for users within network graph**

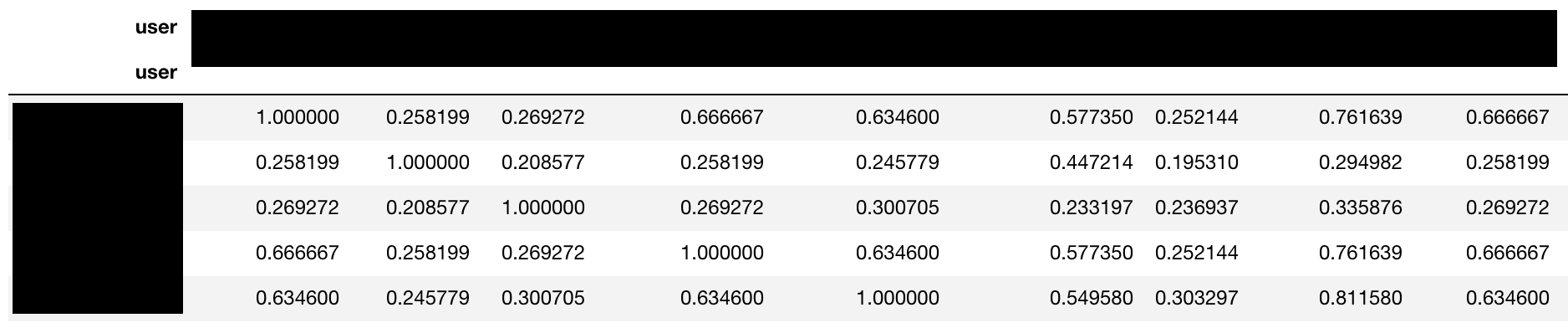
With the data clean and standardized, I used a user-based collaborative filtering recommendation engine to predict preferences for users who are not active users of the Like button. In this section, I will describe the UBCF method and how I applied it to this problem. Also, since I do have data on users who do actively use the Like button, I have the equivalent of ‘in sample’ data. As such, this section will also discuss the potential accuracy of these preference predictions based on this ‘in sample’ data.

In UBCF, we are basically looking for users that are similar to one another based on their reported attributes. In this case, users who went to the same schools, lived in the same places or were born in the same year are considered more similar than users who went different schools, lived in different places or were born in different years. Various distance calculations can be used to measure the similarity between users – in this case, we used cosine similarity. Regardless once user-to-user similarity is calculated, we can then take the dot product between user-to-user similarity and the normalized attributing ratings to arrive at what is essentially a weighted average predicted rating for each attributed for each user.

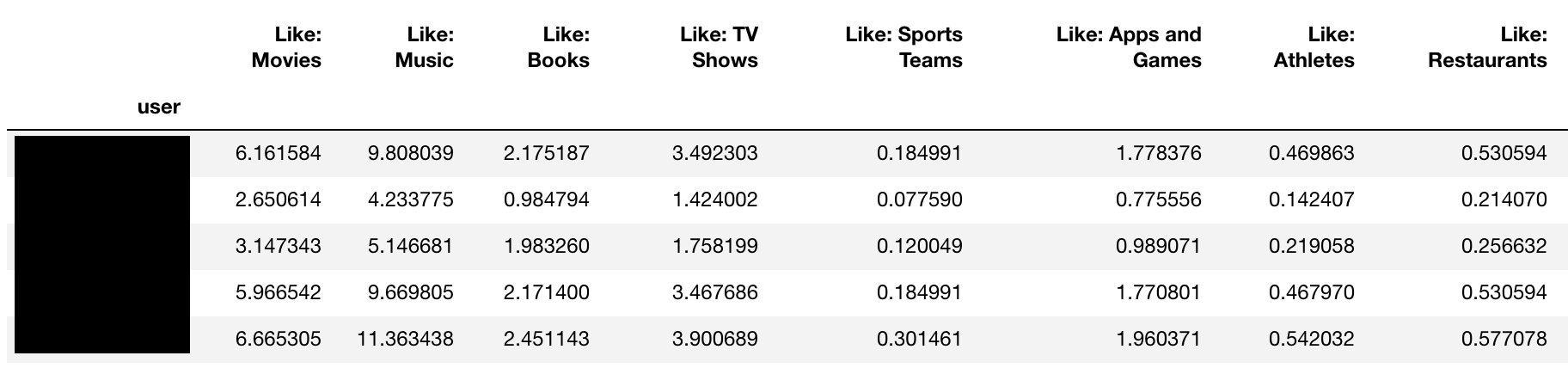
More specifically, we start with a data frame with rows of users, columns of attributes and normalized ratings in the cells as in the snippet below



We then want a user-to-user similarity matrix, so we drop the attributes and replace them with the users, so now we have users in both rows and columns. We then populate the cells with the cosine similarity measurement.



We then take the dot product between the two matrices above to arrive at the prediction matrix, which again has users as rows and attributes as columns (limited to just the attributes we are trying to predict here). But now the cells are populated with the predicted user-attribute preference ratings.



Since we have the actual preference ratings for some users, we are able to calculate some level of accuracy of the predictions being made. In this case, although the number of users reporting preferences for a given attribute varies considerably, for expediency, we will treat all users who have reported at least one preference for any category, i.e. Liked at least one thing, as the “in sample” group. We then look at the correlation between the actual and predicted preference ratings to get some sense of accuracy.

|  |  |
| --- | --- |
| **Attribute** | **Actual-Predicted Correlation** |
| Movies | 0.62 |
| Music | 0.62 |
| Books | 0.65 |
| TV Shows | 0.62 |
| Sports Teams | 0.83 |
| Apps and Games | 0.61 |
| Athletes | 0.56 |
| Restaurants | 0.51 |
| People | 0.94 |

Correlations between actual and predicted values ranged between 0.5 and 0.9, with most in the 0.6 range. These figures are quite conservative since we used the same sample of users for all calculations for expediency, but in reality, not all users reported actual ratings for every attribute. That said, there is certainly room for improvement in the accuracy of these predictions.

**5 Share thoughts on how effective these techniques were in creating personas rich enough for commercial purposes**

Despite the room for improvements in prediction accuracy, we were successfully able to create personas for users that did not report any preference information based on their social networks and the preferences reported within those networks. While the commercial value of the preferences we’ve predicted in this case may be questionable, given the limited time and resources available to this case, I am confident that the approach is sound and can in fact yield rich user personas that will certainly have significant commercial value.

These user personas could have been much richer given more time and resources. For example, using the summarized Like data is just scratching the surface of the Like data itself, not to mention the fact that much more data are readily accessible, e.g. posts, photos, captions, Instagram photos, Instagram hashtags, etc. Services like Spotify and Pandora already use users’ specific artist, song and genre preferences to serve recommended songs. The same could be done for the movies, books, TV shows, etc. Liked by these users. Leveraging posts, photos and other readily available data sources would only broaden the categories of recommendations and strengthen the recommendations within each category.

On the other hand, building a business model so heavily dependent on a single data source, Facebook in this case, is tenable at best. And while this can be somewhat mitigated by leveraging multiple data sources, there tend not to be very many, well established social networks operating at the same point in time given the substantial network effects. That said, the nature of social networks works in our favor in that these platforms take advantage of the fact that their users intrinsically desire to publicly share information about themselves. Similarly, web-scraping is a fundamentally fragile approach to data gathering, especially at the incredible scale necessary to commercialize such a service. Even minor changes to a site could cause the web-scraper to fail, and so such an endeavor would require significant monitoring overhead to keep working. Of course there are examples of successful enterprises that rely heavily on web-scraping, e.g. Mint scrapes data from its users’ banking websites.

Finally, the jury is still out on how best to balance the utility of using social network data for more targeted advertisements vs. the importance of user privacy. Clearly, we’ve swung too far toward monetizing user data without guardrails and need to course correct somewhat. But that said, user data does provide incredible utility. For example, Google Maps and traffic are based on the locations of millions of drivers using Google Maps, and I think the vast majority would agree that we are better off having Google Maps than not. So while this exercise would on the surface seem to be a step backward in protecting user privacy, I would argue that it is less about using user data for commercial purposes and more about being transparent about how we are using it, giving customers the choice to opt-in or out and perhaps most importantly providing true utility back to the user with this data – which means not treating user data as a pure commodity, sold to the highest bidder.