COMP530 - Progress Report
Profile Matching

Erhan Tezcan · Waris Gill · Mandana Bagheri Marzijarani
etezcan19@ku.edu.tr · wgill18@ku.edu.tr · mmarzijarani20@ku.edu.tr

December 13, 2020

1 Original Plan

The original timeline is given in figure 1.

	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
Preparing Dataset	M									
Preparing Pipeline		M, W, E								
Implementing Algorithms			W, E		W, M, E					
Evaluations						W, M, E				
UX Module									E	

Figure 1: GANTT Chart from Proposal.

2 Accomplishments

In this section we discuss our accomplishments and how we achieved them.

2.1 Preparing Dataset (W5-W6 of Figure 1)

For the first step of this project, we started collecting the data to be used in evaluation of our profile matching algorithm. We had originally planned to create a dataset of 1000 Facebook and 1000 Twitter accounts, where 20% of the profiles composed our ground truth.

Before starting to collect the data, the list of Facebook and Twitter users were prepared. We divided our data into two categories: matched and unmatched. To build the list of 200 profiles across Twitter and Facebook that have the same identity, we listed the name of publicly known figures that have confirmed profiles on both platforms. However, we did not want our ground truth to be strongly biased. Thus we kept the number of public figures in our ground truth under 50. The rest of the ground truth set includes regular people who we manually confirmed on both platforms using their image, friends/following list, affiliation, etc.

As for the rest of our dataset, we tried to minimize the possibility of matched profiles. Hence, we selected users on Facebook and Twitter who belong to two completely different social circles. For Twitter, we selected 800 people with cyber-security, IT and academic backgrounds. For Facebook we started by selecting people who worked in an auto dealership in Los Angeles and collected the usernames of their friends.

To collect the selected users' publicly available data, our initial plan (as described in the project proposal) was to use Twitter's developer API and Facebook's Graph API. It is worthy to mention that we took Dr. Emre hoca's advice seriously and investigated the practicality of our approach before proposing it. While we submitted the proposal, our application to open a Twitter developer account was under review and we had already submitted a simple Facebook App for review. We had also investigated the daily rate limit of both APIs and their workarounds. Unfortunately our request to create a Twitter developer account was rejected three times, despite trying with the university email account and explaining thoroughly and exactly what we meant to use the data for.

As an alternative, we decided to look for publicly available online Facebook/Twitter data scrappers. There are many projects available on GitHub and we tried a few of them. To the best of our knowledge there were no Facebook/Twitter scrapper that were both correct and up to date. This is most likely because both platforms regularly change their page structures and technologies and the crawlers are usually outdated.

Consequently, we developed a Facebook and Twitter scrapper using Python and Selenium[1], coupled with jQuery selectors. The scrapper opens up Twitter and Facebook, logs into them, then iterates over desired profiles and several pages of each profile. On each page, the scrapper injects jQuery into the page, then uses jQuery selectors and helpers to select the desired data and return it as a dictionary. These dictionaries are then combined and saved as JSON files. Additionally, several pages required scrolling down dozens of times and waiting for more data to be loaded (e.g., friends list on Facebook and Twitter).

Furthermore, a particularly challenging issue was the API rate limits available on both platforms. For a typical profile, we would need to send 20 to 100 requests to the platform. Not only is sending these many requests slow, it also typically results in the account and/or the IP address being rate limited and further access denied for an unspecified amount of time.

The entirety of each profile's information was collected in two steps. In the first step, we collected the general information of the profile on both platforms, sans the friends list for Facebook and followers/followings list for Twitter. In this first step, we were able to work around platforms' rate limiting using VPNs and changing IPs every 20-40 profiles, before the

[1] https://www.selenium.dev/

platform would block our IP for unusual activities.

In step two, for a given profile we had to first load the friend, follower and following list pages, then scroll down multiple times (each scroll would load about 5 to 20 new friends), and then extract the usernames of the friends list. In the case of Facebook, we were able to collect the friend list using VPNs, as Facebook did not flag the account we used for logging in and collecting the data (only the IP address was flagged, and a sufficiently diverse VPN was enough of a workaround for the rate limit). Even though the process was extraordinarily slow, we were able to collect the list of the first 100 friends of about 1000 Facebook profiles.

In the case of Twitter, contrary to profile information, to collect the follower/following list, authentication with a valid account was required. Furthermore, Twitter employs several mechanisms to prevent logging in automatically. As such, the login step had to be done manually each time the scrapper was launched. Twitter repeatedly suspended the accounts we used for logging in. To circumvent this limitation, every time we used a new VPN connection and a new account, which would get suspended after scrapping a few additional profiles.

Another issue with Twitter was that after loading a follower list and scrolling down to get the entirety of the list loaded, the items higher on the list would be removed from the page and disappear. Given the aforementioned challenges with Twitter, the only data that we were **not** able to successfully collect was the follower/following list.

After collecting about 1950 profile's information across Facebook and Twitter including almost 20 percent matched profiles, we imported the data into a MongoDB database hosted on a Vultr[2] cloud server. Our database has two collections, Facebook and Twitter, each of which is indexed by username as id.

As a further note regarding this task, our team member Mandana is still exploring to find a workaround for collecting the follower's list using a twitter App called Vicinitas[3].

2.2 Preparing Pipeline (W6-W7 from Figure 1)

In line with the previous task, this "preparation" task is regarding the question: what type of data can we obtain and how can we use it? Due to not being able to use an API and use a crawler instead (as described above), the path of this task differed greatly, from "reading API docs and preparing the code in compliance" to "looking at what we obtain by using a crawler and using that".

We have two modules to note in this task, implemented as classes in our Python code:

[2]https://www.vultr.com/

[3]https://www.vicinitas.io/

- **mongo.py**: This is a simple API that connects our script to the MongoDB server which holds the Twitter and Facebook users. It also implements a basic post-processing on the document, such as converting the crawled date strings into date objects and combining several fields into a single string, such as in the case of 3 "education" fields from Facebook. The API is also served to the Matcher (which we next describe) in case of any on-demand requests to DB.

- **matcher.py**: This module is essentially the heart of the project. It has two main functions that it exposes to outside: one to match a Twitter user to Facebook user, and the other to match a Facebook user to Twitter user. The algorithms are and further will be implemented within this module.

2.3 Implementing Algorithms (W7-W10 from Figure 1)

We have currently implemented a few algorithms that may serve as building-blocks for the further iterations of the project. We have used the transformers[4] package for NLP tasks 1, 2, 3. This package is chosen because it allows us to use state-of-the-art NLP algorithms. For the image processing task we have used skimage's[5] Structural Similarity Index function only, however we will implement a more advanced algorithm for this purpose in the coming days.

1. **Text Summarization**: Summarizing tweets, headlines, biography etc. might reveal some important information from these texts. Pruning the redundant information from such texts will ease the effort of matching them from one domain to another.

2. **Text Semantics**: Tweets, headline and biography may hold semantic information that may be useful to us. We are still in the thought process for this, however we thought we may eventually use it.

3. **Text Similarity**: One of the most required algorithm for us is the text similarity, where we can see how similar a text is to one another. We could be comparing biographies or location information in this way.

4. **Image Similarity**: After obtaining a certain set of candidates from one domain for a user of the other domain, we may compare their profile images and see if there is a considerable similarity among them.

[4]https://huggingface.co/transformers/
[5]https://scikit-image.org/

3 Remaining Tasks

In this section, we discuss the tasks on which we are working on.

3.1 Algorithms

We still have not connected the algorithms in a manner that can match two users from different domains. That is the main remaining task, and it also requires some more advanced algorithms than what have done so far. We expect to stick to the revised timeline in the completion of this task.

3.2 Evaluations

As written in the previous subsection, since we have not done any matches so far, and the evaluations are depending on actually having a match, we are yet to begin this task.

One caveat here is that we do not know how our algorithm may perform beforehand, so if the evaluation turns out too bad, depending on the time, it may too late to fix some things. However, we still find value in it, as this would indicate how some algorithms are **not** suited for this kind of applications. We expect to follow the revised timeline in the completion of this task.

3.3 UX Improvement

We have still not started the UX improvement task, as it is the final one. To rewrite what this task was about: basically instead of console outputs we are planning to generate an HTML output, so that we can see how a user from one domain was matched to the other, via which information and so on. As stated in our proposal, we are leaving this as the final task. We expect to follow the revised timeline and leave this task as the last.

4 Self-Assessments and Planning

The major setback we had was due to not being able to use Twitter or Facebook APIs. Our developer accounts were rejected (and this happened days after our proposal), so instead of waiting and hoping for other ways to get developer access we had to get going and collect data! The way to do that boiled down to using a crawler. Using a crawler is significantly "dirtier" than using an API to get what we want, so this put additional stress on the first and foremost task. Further problems were described in section 2, so we will not be reiterating them here, however we would like to stress again that the major timeline shift happened due to the unexpected rejections from Twitter and Facebook.

We are 1 week behind our planned schedule, but we are catching up to it. We were also generous with the last UX task in the original timeline, so shifting all others 1 week forwards will not put much pressure on that task.

As also shown in the revised timeline, the algorithm implementation and evaluation will go hand-to-hand as much as it is possible, to catch up to our original timeline.

All in all, we are currently behind but we are ready and on our way to catch up!

5 Revised Timeline

The revised timeline is given in figure 2.

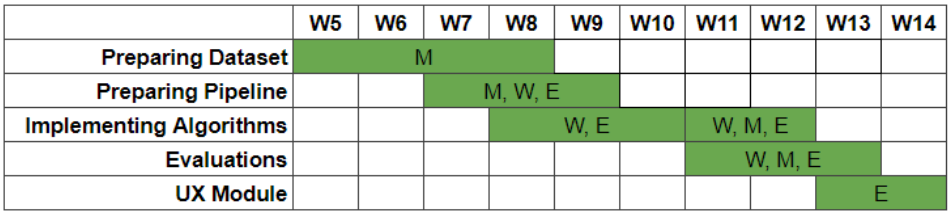

Figure 2: New GANTT Chart. Note that at the time of this submission, we are ending week 10, shown as **W10**.

There are two important things to mention here:

1. **The first task is given an extra 2 weeks, and the whole timeline is pushed 1 week forward.** Therefore, we are automatically 1 week behind of our original timeline, however, on time with our revised timeline.

2. **The "implementing algorithms" task is extended a week.** Since we are going to try few approaches, we thought we can improve some aspects of the algorithms while preparing the evaluations of the completed parts in parallel. As a result, the former 2 week difference (W11, W12) between the ending times of evaluations and implementations, is now reduced to 1 week (W13).

6 Changes to Project

As discussed in section 2, Twitter or Facebook API was not an option anymore and we had to use a crawler. We had written our proposal towards using an API, and since we used a crawler instead we felt like this should also be noted down under this section.

Appendix

The current source code is included in the submission file. It is a direct release export from our private GitHub repository.