TWITTER DASHBOARD:

A WEB SERVICE AGAINST ONLINE HARASSMENT

A Thesis Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Information Systems

by

Jingxuan SUN

May 2020

ABSTRACT

Political discussion on major social media platforms such as Twitter is often flooded with conflicts and polarization. Users sometimes would use adversarial expressions towards political candidates to undermine their legitimacy or intend to discourage them from competing. Thus, identifying whether the interaction is adversarial between a reply and a tweet and whether the content is direct to the political candidate is essential to step towards a methodical and harmonious online environment. We focus on the direction of adversary observed in the tweets from 2018 US general election period, produced well-formatted datasets which contains more than 1.5 million data points covering tweets, user information and candidate information, and developed multiple models combining heuristics and machine learning techniques to predict adversarial direction.

Continuing with last semester's harassment direction model development, we extended our work to embed the model into the backend of a web service - Twitter Dashboard, in order to help registered users automatically filter adversarial content from his/her Twitter account. We built the web client with Flask framework on Google Cloud Platform. On the server side, we modified the models from direction classification to predicting whether to mute a replier, using logistic regression and BERT models. Users also have the freedom to check muted replies and choose to unmute certain repliers. User tests received satisfactory model performance.

BIOGRAPHICAL SKETCH

Jingxuan SUN comes from a CS background and is a master student in Information Science at Cornell Tech. She has genuine interest in observing human behavior and extracting insights from data. She has been working in the data science team at Kabbage Inc. as an intern, working PoC projects on leveraging deep learning for predicting users' risk of default at loan application time. Jingxuan's work has the focus on using machine learning knowledge to find fun or mind-blowing facts, aesthetically visualize them, and make interesting products. She previously served as user researcher for Cornell Tech Product Studio team, carrying out the tasks including user interview, contextual inquiry, AB experiments.

Xiran SUN majored in Physics during her undergraduate and is now a second year Information System master student studying at Cornell Tech, in the track of connective media. She has experience in full-stack development during her previous internship at Goldman Sachs. Recently, she found her interests in the area of Virtual Reality and Augmented Reality development. She has a VR project which could teach users how to write Chinese Characters with a VR headset on. Personal interests include jogging, playing badminton, and dancing.

ACKNOWLEDGEMENT

We would like to thank our thesis advisor Professor Mor Naaman of the Department of Information Science at Cornell Tech. Professor Naaman always has foresight in pointing out possible directions whenever current methods reach a bottleneck. He is also very approachable and patient with all our questions with genuine care.

We would also like to thank Yiqing Hua, PhD of Computer Science at Cornell Tech. Yiqing worked with us closely and spent much time on the whole project. We are very lucky to have her for general advice on technical difficulties, as well as the consultation on implementation details. We are gratefully indebted to her for the very valuable comments on this thesis.

We would also like to thank fellow students who helped us with data labeling and offered suggestions on hybrid model architecture setup. The models would only be an empty shell without data and concrete training.

CHAPTER 1

INTRODUCTION

Social media platforms provide modern people a safe harbour to express ourselves freely. For example, users could directly reply to a political candidate to discuss an issue or convey their personal opinions on Twitter. However, the content users reply to a tweet sometimes can be not friendly at all. Political candidates are even more attractive to and vulnerable of online harassment due to drastic opinion conflicts and potential consequences resulted from misinformation or disinformation.

There has been a great number of works focusing on tweet sentiment analysis, but the problem is more complex in a political context. First, the semantics of the short text becomes more subtle and contextual. Second, it is hard to identify the direction of the adversarial sentiment due to the platform's functionality. Sometimes even though users reply to a tweet, it doesn't mean that they are directly targeting the candidate who posted it.

Last semester's work focused on an automatic approach of detecting the direction of adversary in political interactions on Twitter. We processed 1.7 million tweets posted by general users in reply to or mentioning candidates in the 2018 US general election, whose toxicity were measured by Perspective API [3]. After selection, those tweets containing adversary content were assigned a binary label indicating whether the direction was toward the posting candidate. Finally, a hybrid model combining the advantages of deep learning and heuristic features was created for automatic classification.

In this semester, we diverged from model improvement to embedding them into software applications in order to make real world impact. We prototyped the web based solution in a fast and dirty approach to offer users who connect to our platform the “hand-free” service of automatic content and account filtering. Each user has a personalized back end model tailored towards his or her own definition of harassment. We also give users the freedom of controlling the filtering process by whitelisting accounts that are already labeled as offensive by our models.

CHAPTER 2

RELATED WORK

2.1 Towards Measuring Adversarial Twitter Interactions against Candidates in the US Midterm Elections [2]

In this paper, Hua et al. proposed a technique called “directionality via party preference (DPP)” that could better quantify explicit adversarial interactions towards candidates. It comes with two heuristics, 1) the tweet’s author leans towards the political party opposing that of the candidate; 2) the tweet uses second person pronoun. By applying these to tweets posted by popular candidates, an algorithm that can discover target-specific adversarial lexicons are introduced.

2.2 Online Harassment Campaign: Tweets Adversary Direction Detection [4]

This paper we wrote last semester offers a brief overview of state-of-the-art approach in sentiment analysis on short text and the technologies used for harassment direction detection. A hybrid model was developed and tested. It incorporated heuristic features mentioned in Section 2.1, an LSTM trained on word embeddings retrieved from our own dataset and served as the “black-box” component of the hybrid model, as well as complementary features unique to tweets such as hashtag and emoji usage. The final model reached 80.4% accuracy in direction detection.

CHAPTER 3

IMPLEMENTATION - SERVICE ARCHITECTURE OVERVIEW

In this project, we used App Engine, Data Store and Compute engine provided by the Google Cloud Platform [1] to help us build the platform. This project contains three main parts, front-end part, database part, and a server part which contains the machine learning model we are using.

For the front-end part, which we implemented in App Engine from GCP, it can handle user interactions, once a customer registers for our platform and authenticate themselves, we can start getting the tweets information from the user, and start "listening" to their new tweets, and the information of the tweets will be stored into our database. The database we are using is also from GCP named DataStore, which can be accessed both from the frontend and the server. The server contains a machine learning model that we are using to make decisions whether to mute users for our customers based on the information stored in the database. Here is a diagram of how these three parts are linked with each other.

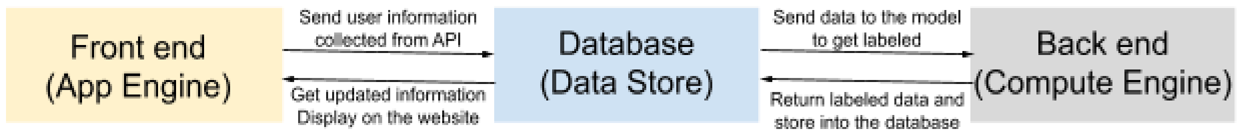

Front end (App Engine)
Send user information collected from API
Get updated information Display on the website
Database (Data Store)
Send data to the model to get labeled
Return labeled data and store into the database
Back end (Compute Engine)

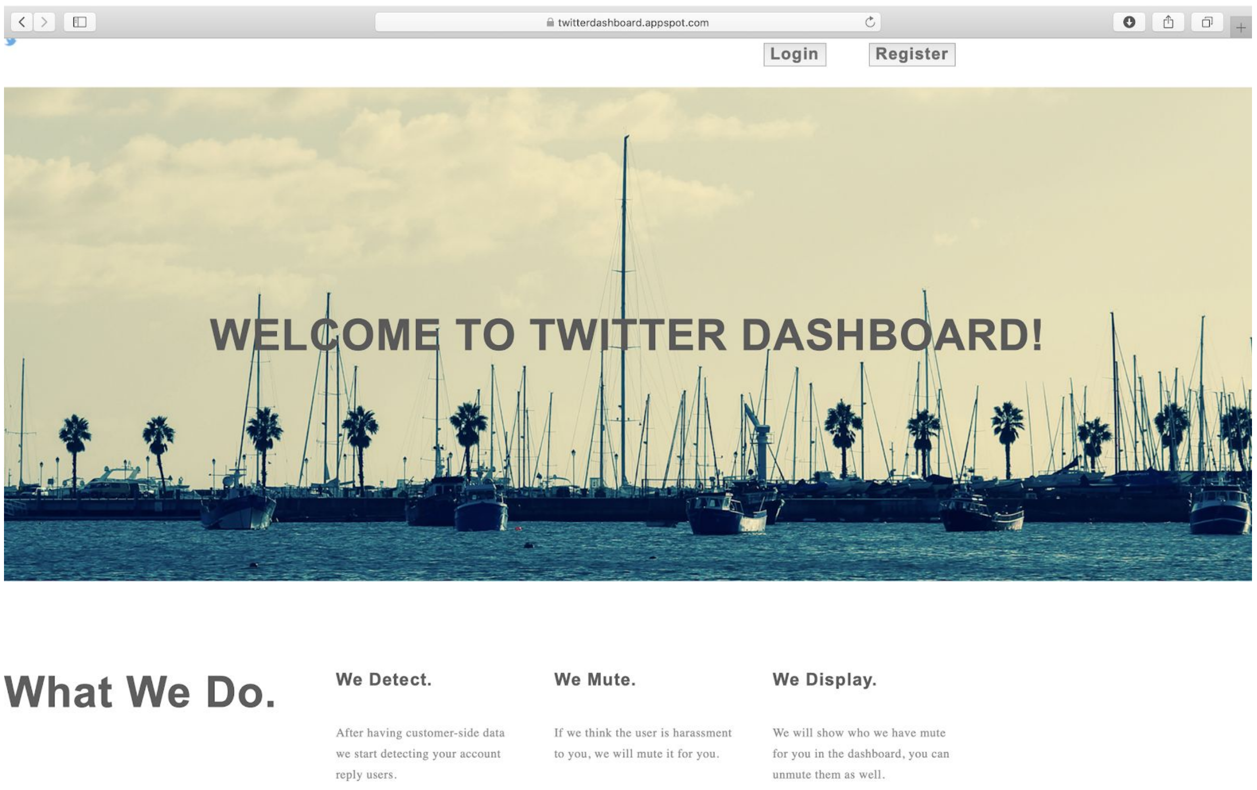
Figure 1: Service Architecture Overview

3.1 Front End (App Engine)

The web service architecture is built with Python Flask framework. Client-facing interaction components are written in HTML and CSS under Jinja2 framework. The final prototype application is deployed on Google App Engine hosted by Google Cloud Platform, where the backbone of the service is conveniently connected to Google Cloud DataStore for storing raw tweets, replies and user information streamed from Twitter APIs [5], as well as our predictive models hosted on Google Compute Engine instances. Both the light-weight web framework and the seamless integration of the platform functionalities offered us great simplicity in binding different modules of the service.

The following section introduces the application logic behind each web page visible to our end users.

Index: On entering our dashboard, the user is prompted with choices of "login" or "register". Those who are coming for the service for the first time should choose "register".



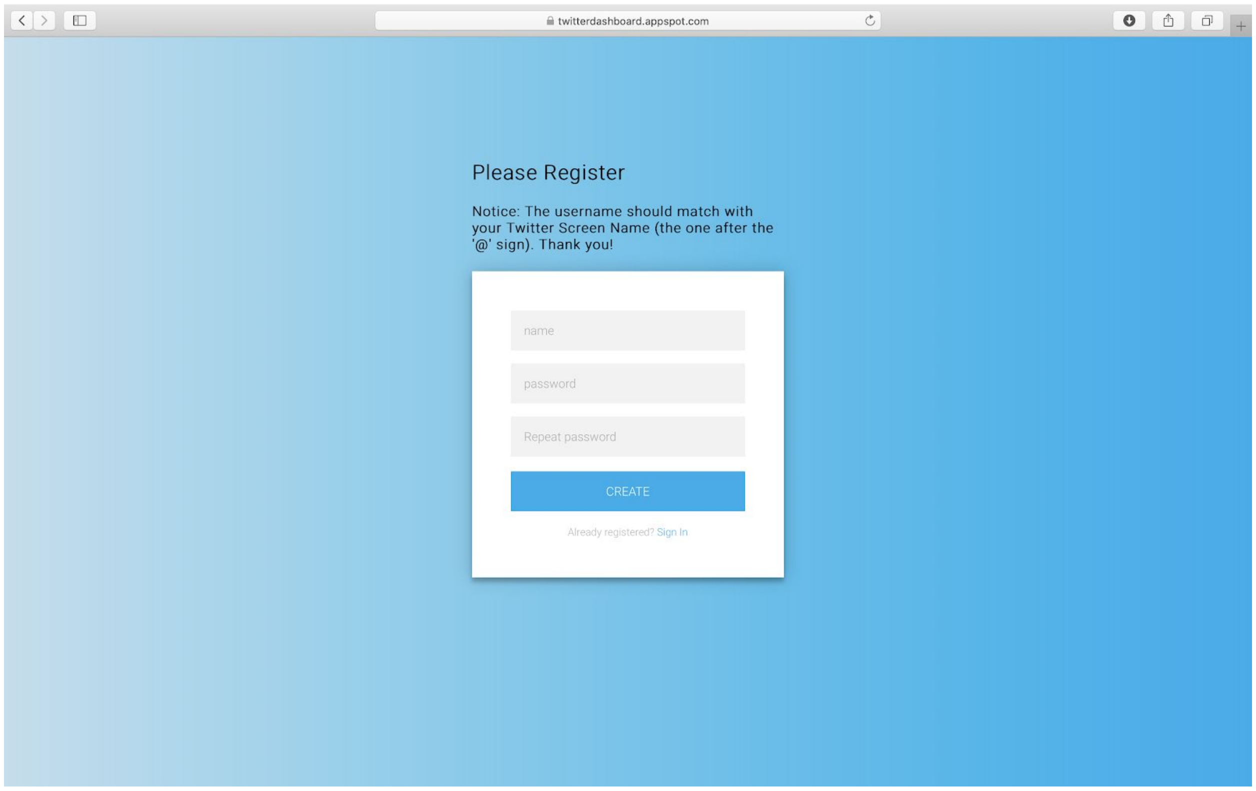
After having customer-side data we start detecting your account reply users.

If we think the user is harassment to you, we will mute it for you.

We will show who we have mute for you in the dashboard, you can unmute them as well.

Figure 2: Index Page

Register: The app redirects the user to "register" page, where he is required to fill-in a username (which is the same as the user's Twitter account screen name), a password and a confirmation of the password. Clicking on "submit" will redirect him to the Twitter authentication page.



Please Register

Notice: The username should match with your Twitter Screen Name (the one after the '@' sign). Thank you!

Figure 3: Register Page

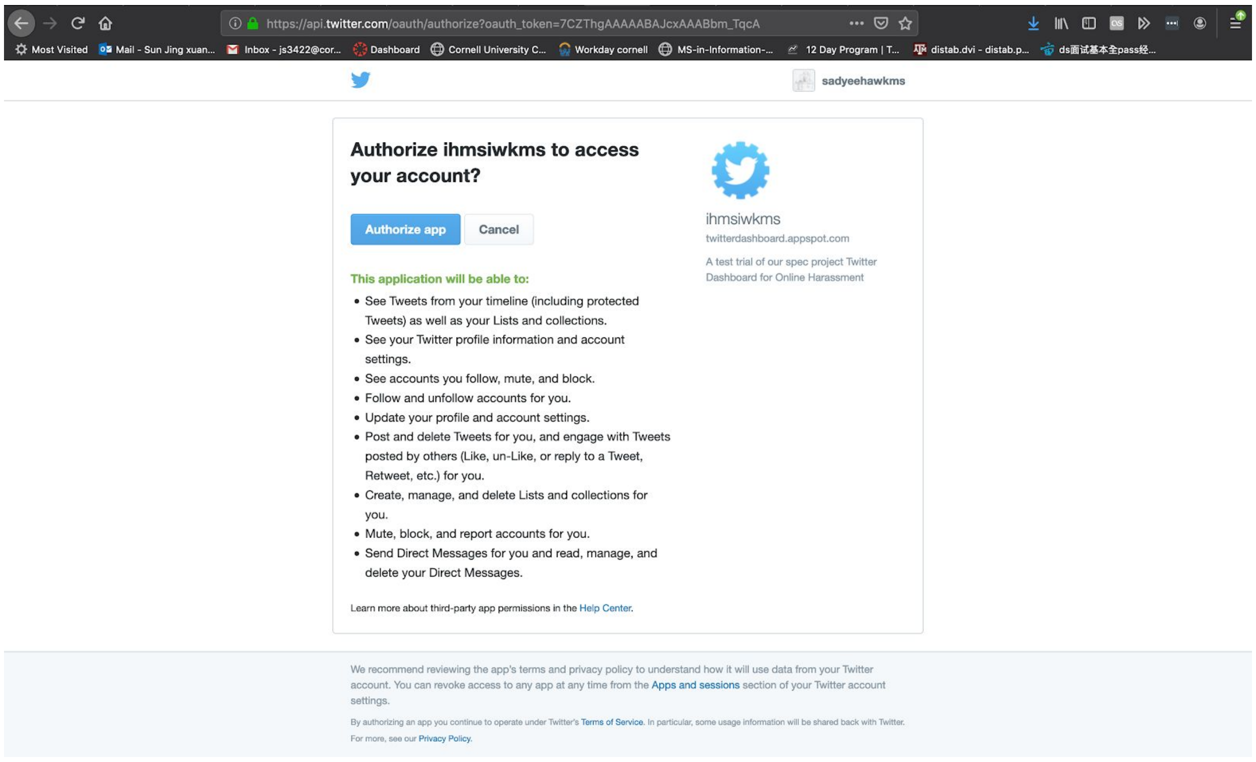
Auth: In authentication function, an OAuth request to Twitter is set up with a set of customer public/secret keys, which are obtained from Twitter Developer account. Our app redirects user to the Twitter authentication page, where he can log into his Twitter account and click to approve our access.

Callback: On approval, OAuth redirects the user back to a callback url, from which our app can retrieve a set of access public/secret keys. These two pairs of keys are further used to connect and get access to the user's Twitter content and activities. This step is invisible to users.

In addition, a new user's register process completes at this step. On the backend, the success of authentication triggers the system to store user's login information to our database.

More importantly, we connect our service to Twitter Streaming API in the background to start continuously listening to user's activities, including posting of new tweets, receiving new replies, muting other users. On capturing these actions, the content is continuously streamed and stored in our database for further processing.

The user will be redirected to the initial labeling page.



https://api.twitter.com/oauth/authorize?oauth_token=7CZThgAAAAABAJcxAAABbm_TqcA

Most Visited | Mail - Sun Jing xuan... | Inbox - js3422@cor... | Dashboard | Cornell University C... | Workday cornell | MS-in-Information-... | 12 Day Program | T... | distab.dvi - distab.p... | ds面试基本全pass经...

sadyeehawkms

Authorize ihmsiwkms to access your account?

Authorize app | Cancel

ihmsiwkms
twitterdashboard.appspot.com
A test trial of our spec project Twitter Dashboard for Online Harassment

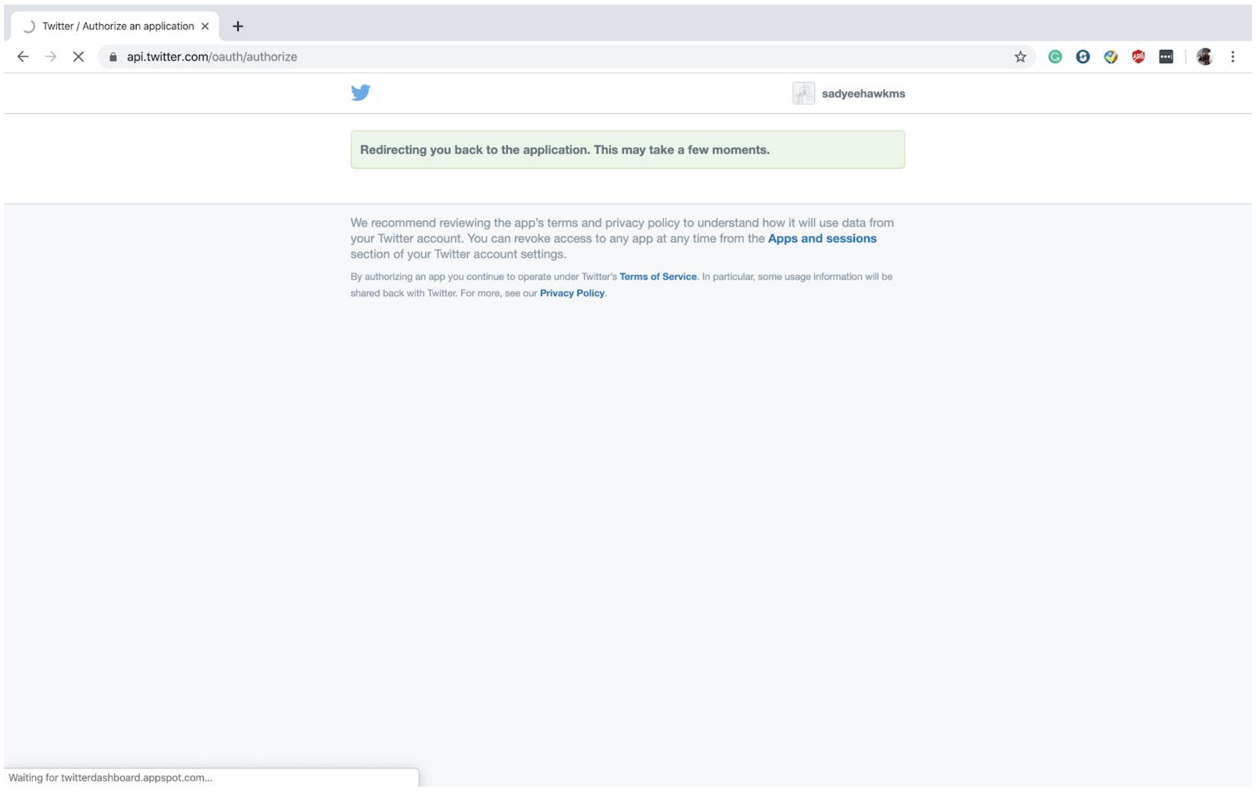
This application will be able to:

- See Tweets from your timeline (including protected Tweets) as well as your Lists and collections.
- See your Twitter profile information and account settings.
- See accounts you follow, mute, and block.
- Follow and unfollow accounts for you.
- Update your profile and account settings.
- Post and delete Tweets for you, and engage with Tweets posted by others (Like, un-Like, or reply to a Tweet, Retweet, etc.) for you.
- Create, manage, and delete Lists and collections for you.
- Mute, block, and report accounts for you.
- Send Direct Messages for you and read, manage, and delete your Direct Messages.

Learn more about third-party app permissions in the Help Center.

We recommend reviewing the app's terms and privacy policy to understand how it will use data from your Twitter account. You can revoke access to any app at any time from the Apps and sessions section of your Twitter account settings.

By authorizing an app you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.


Twitter / Authorize an application

api.twitter.com/oauth/authorize

sadyeehawkms

Redirecting you back to the application. This may take a few moments.

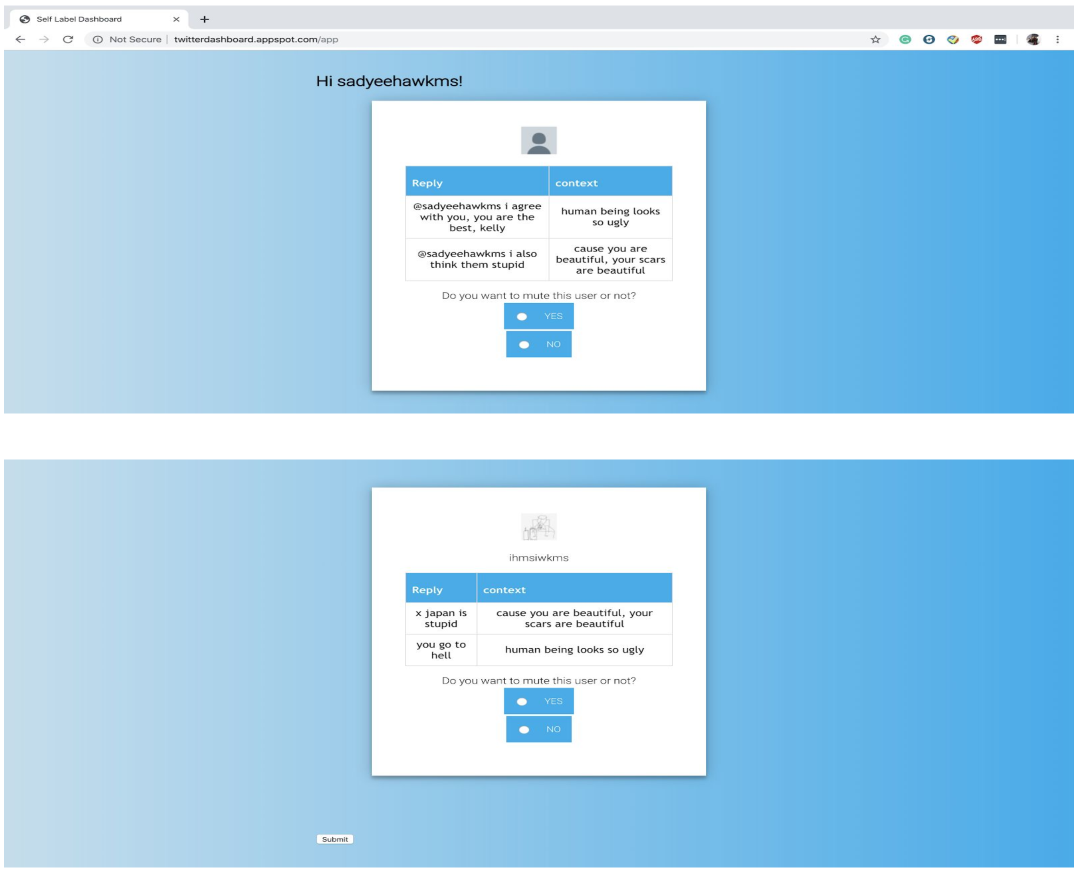
We recommend reviewing the app's terms and privacy policy to understand how it will use data from your Twitter account. You can revoke access to any app at any time from the **Apps and sessions** section of your Twitter account settings.

By authorizing an app you continue to operate under Twitter's **Terms of Service**. In particular, some usage information will be shared back with Twitter. For more, see our **Privacy Policy**.

Waiting for twitterdashboard.appspot.com...

Figure 4: Twitter Authentication

Init: In order to get initial training data for fine-tuning our backend personalized models, we prepare an initial labeling process for each new coming user. This page displays a series of other users who have replied to our client's tweets. For each replier, we use Twitter Search API to collect and filter from all the replies he posted a balanced sample in terms of toxicity score. Toxicity score is calculated from Perspective API, reflecting the degree of harassment and adversary. We also managed to balance the number of replies displayed from each replier to free clients from heavy labeling tasks. As we group the replies according to repliers, the client's labeling task is to decide whether the system should mute certain repliers. The app collects the user's response and saves the label together with the tweet-reply text pair into the database for further processing. At the same time, we carry out an initial search for the user's mute list, containing other users that have already been muted or blocked prior to connecting to our dashboard. These actions are all carried out before the user is redirected to the dashboard.



Hi sadyeehawkms!

Reply	context
@sadyeehawkms i agree with you, you are the best, kelly	human being looks so ugly
@sadyeehawkms i also think them stupid	cause you are beautiful, your scars are beautiful

Do you want to mute this user or not?

ihmsiwkms

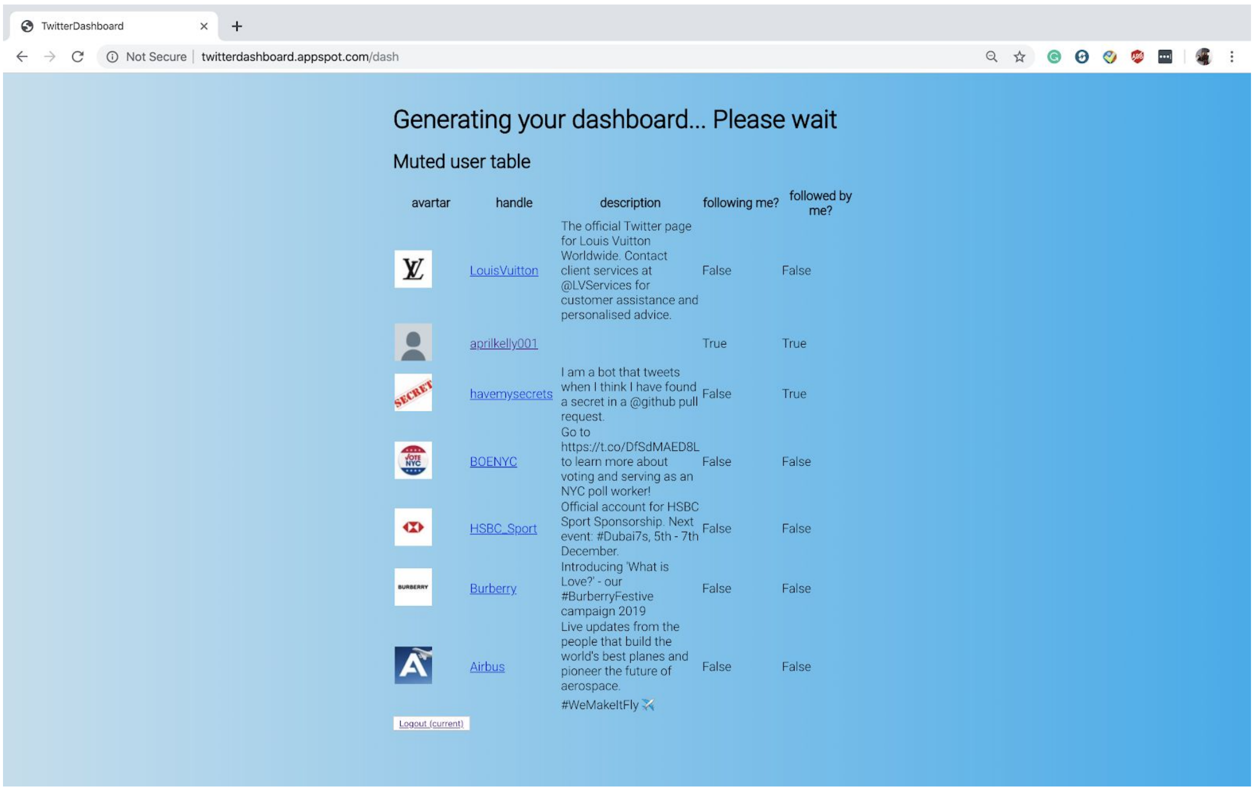
Reply	context
x japan is stupid	cause you are beautiful, your scars are beautiful
you go to hell	human being looks so ugly

Do you want to mute this user or not?

Figure 5: Self-Labeling Page

Dash: For the prototype application, dashboard page displays the information of the list of accounts that our service helped the client to mute, depending on the character of the replies from that user. Information includes the user's avatar, handle, description and network information (whether the muted user follows the client and vice versa). Some of these attributes are also used as features in our predictive models, so that offering them here is more transparent to the client. The content on this page is essentially the output of our models.

This step completes the client's activity cycle.



Generating your dashboard... Please wait

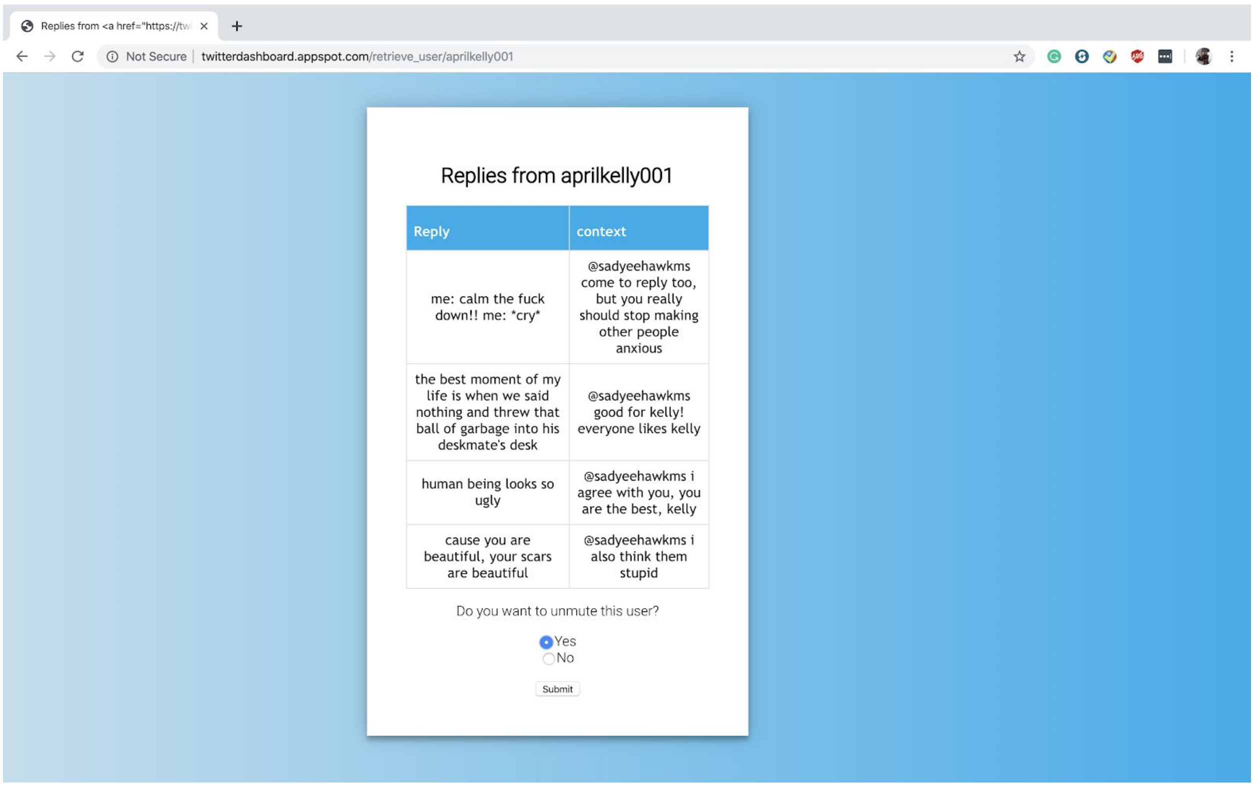
Muted user table

avartar	handle	description	following me?	followed by me?
	LouisVuitton	The official Twitter page for Louis Vuitton Worldwide. Contact client services at @LVServices for customer assistance and personalised advice.	False	False
	aprilkelly001		True	True
	havemysecrets	I am a bot that tweets when I think I have found a secret in a @github pull request.	False	True
	BOENYC	Go to https://t.co/DfSdMAED8L to learn more about voting and serving as an NYC poll worker!	False	False
	HSBC_Sport	Official account for HSBC Sport Sponsorship. Next event: #Dubai7s, 5th - 7th December.	False	False
	Burberry	Introducing 'What is Love?' - our #BurberryFestive campaign 2019	False	False
	Airbus	Live updates from the people that build the world's best planes and pioneer the future of aerospace. #WeMakeItFly ✈️	False	False

Logout (current)

Figure 6: Dashboard Page

Retrieve user: An optional step is that our user can go into the detailed reply records from a certain replier. This gives the user more freedom to be more autonomous in choosing whether to mute the replier. This also serves as a proactive way to personalize the back end models, tailored towards each user's preference. For example, a user might be more sensitive to sexual harassment but feels indifferent with what classified by Perspective API to be insulting. After telling us his/her preference, the model would be able to capture this characteristic and focus learning more on muting sexual harassment contents.



Replies from <a href="https://tw

Not Secure | twitterdashboard.appspot.com/retrieve_user/aprilkelly001

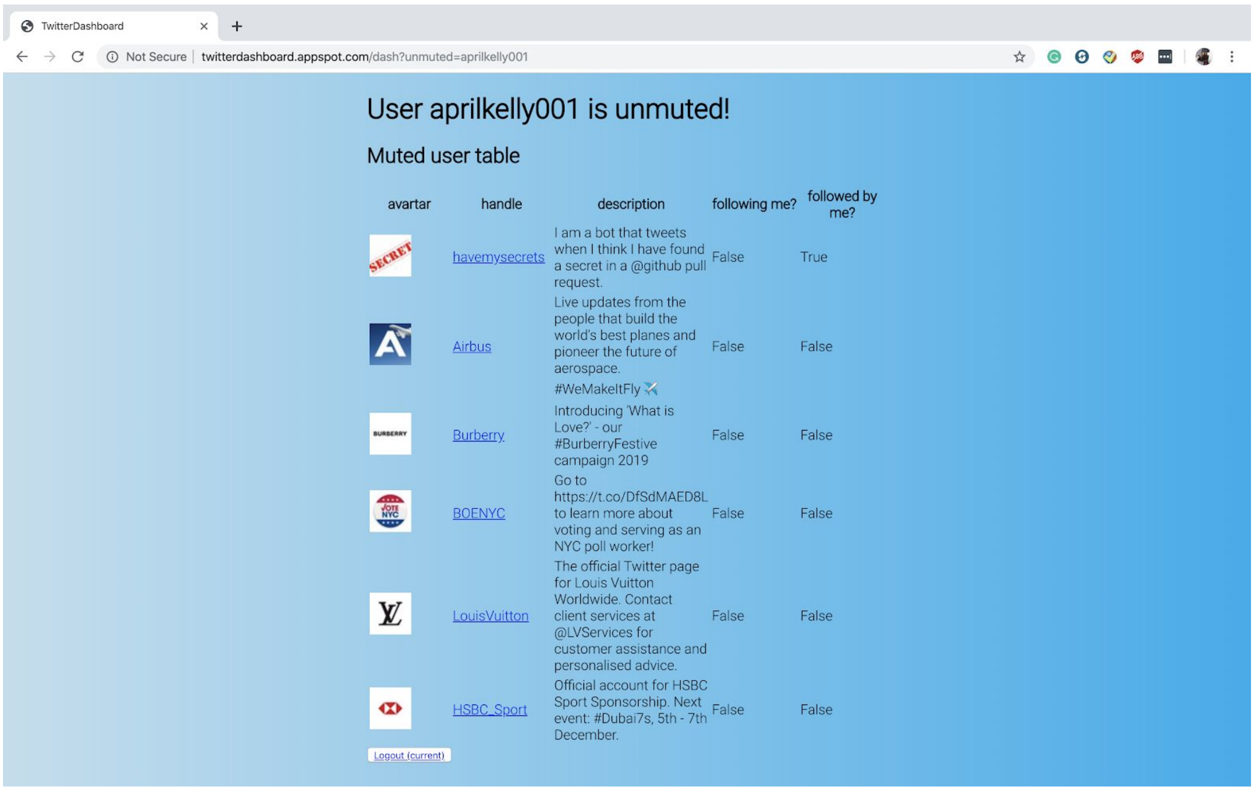
Replies from aprilkelly001

Reply	context
me: calm the fuck down!! me: *cry*	@sadyeehawkms come to reply too, but you really should stop making other people anxious
the best moment of my life is when we said nothing and threw that ball of garbage into his deskmate's desk	@sadyeehawkms good for kelly! everyone likes kelly
human being looks so ugly	@sadyeehawkms i agree with you, you are the best, kelly
cause you are beautiful, your scars are beautiful	@sadyeehawkms i also think them stupid

Do you want to unmute this user?

[✓ Yes]
No

Submit



TwitterDashboard

Not Secure | twitterdashboard.appspot.com/dash?unmuted=aprilkelly001

User aprilkelly001 is unmuted!

Muted user table

avartar	handle	description	following me?	followed by me?
	havemysecrets	I am a bot that tweets when I think I have found a secret in a @github pull request.	False	True
	Airbus	Live updates from the people that build the world's best planes and pioneer the future of aerospace. #WeMakeItFly ✈️	False	False
	Burberry	Introducing 'What is Love?' - our #BurberryFestive campaign 2019	False	False
	BOENYC	Go to https://t.co/DfSdMAED8L to learn more about voting and serving as an NYC poll worker!	False	False
	LouisVuitton	The official Twitter page for Louis Vuitton Worldwide. Contact client services at @LVServices for customer assistance and personalised advice.	False	False
	HSBC_Sport	Official account for HSBC Sport Sponsorship. Next event: #Dubai7s, 5th - 7th December.	False	False

Logout (current)

Figure 7: User has the freedom to control whether to mute certain replier.

Login/logout: For returning users who have already finished the register, authentication and Twitter API setup process, he can simply login from the index page and logout from dashboard page. The system will verify his login information in the background and retrieve predictions from database for display.

3.2 Database (DataStore)

Here is an overview of how we manage the information stored in the database. In general, there are three types of tables we used in our project.

- User
- Bm
- [id]
- Mock, candidate tables

The first one is the basic user information table *users* that stores the user's name, access tokens, user password as well as their Twitter account ID. Once the user registers into our system, we will store the above information into the user table.

As the user registers to our website, we start to use the Twitter streaming API to record the tweets that users post and the replies they get. In this case, we automatically generate a table exclusively for each user named with his/her Twitter account ID, in which contains two kinds of data -- tweets that the user posts and replies that we get for each tweet. Thanks to the capability of datastore to deal with unstructured data, we can put them in one table. For the tweets data part, we collect the tweet ID, tweet text and set the tweet ID as the key to facilitate future inquiries. For the replies data part, we collect the corresponding context id, reply id, reply username as well as some features we get from Perspective API including toxicity level, threat level, insult level and so forth, which are used as a feature in the machine learning model.

In the third table *bm*, we store a map between each registered user and their muted user list, identifying by their twitter user id. We keep updating this table by the results given by our machine learning model and display thee result for each user in the front end dashboard.

A complementary table is mock, containing mock data to be further used in supplementing back end model training process. This usage will be discussed in detail in model training section. The structure of each record is similar to that of the replies stored in each user's exclusive table, containing reply id, reply username, reply user id, reply text, context id, reply to id, reply to name, context. We also manually replicated the same user table structure for several political candidates and treated them as our potential clients. The data also comes from the streamed tweets from last semester.

3.3 Back End (Compute Engine)

3.3.1 Functionality of models and their relationship with app logic

The predictive models are hosted on Compute Engine virtual machine instance. Each user will have their own personalized model, trained from their account content. More specifically, all models share the same architecture but weights are tailored toward each user's content and preference.

After initial connection to Twitter Streaming API, as content flows in, a recurrent cron job is set up to train the models with a fixed interval (we typically set up the training once a week). Compute Engine instance is linked with Datastore so that we are able to retrieve raw training data and labels from the database. Tensor formation and training is done locally, and model weights are stored in the instance.

The queries are carried out in two fashions. First, similar to training, we also set up batch querying tasks once a week to predict on additional repliers collected within the week. Instead of doing all queries on new repliers when client requests to show the dashboard, this is out of the consideration of minimizing the dashboard display latency. Second, if client requests to use the service between two scheduled querying tasks, the prediction is also carried out on the fly for any new repliers. All labels are stored back to the database and retrieved from there upon request.

CHAPTER 4

IMPLEMENTATION - MODEL ARCHITECTURE

As the objective of the model has diverged from the last semester, we have designed two new models.

The classifier answers the question of whether a harassing reply is directed toward the original tweet poster and the unit of analysis is one tweet-reply pair. For the background model of the dashboard, as we want to help users to mute offensive people, we would want to analyze the text aggregated on user level. The label has become whether to mute a user.

The full model has a complex architecture. The replies from each user are first passed through a pre-trained BERT [5] model embedding layers in order to be matched with their word embedding vectors. The vectors of adversarial replies are then passed through the direction model we developed last semester to obtain the direction labels. The normalized count of directing replies is used as a third feature. A mean tweet embedding vector from BERT embedding layers is augmented with these hand-picked features, and the newly formed vector is then passed through the rest layers of the BERT model.

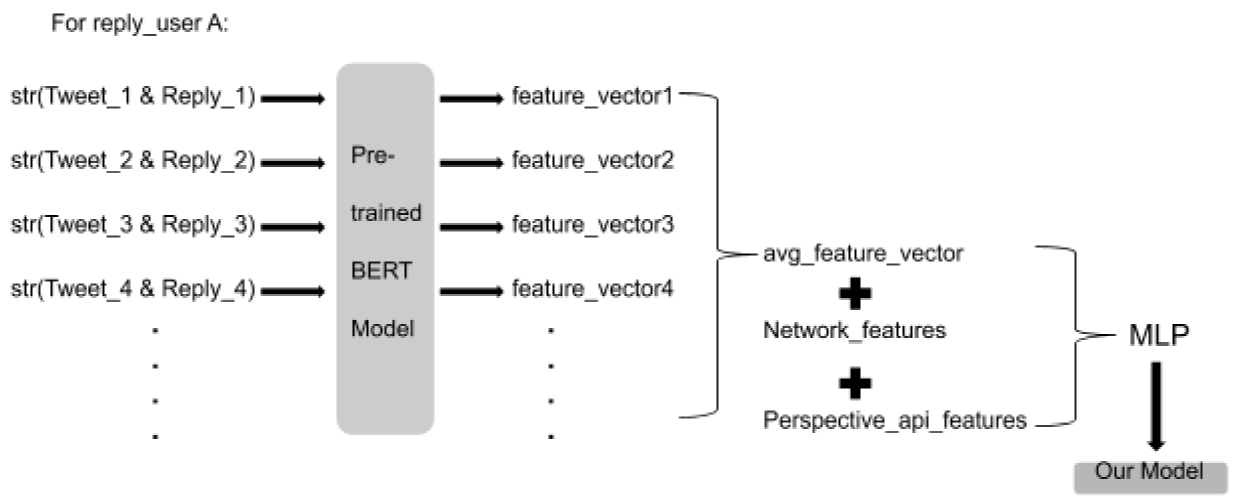


Figure 8: Full Model Setup

During the fast prototyping phase, we created simple logistic regression models to serve as an approximation of the deep learning models and to get insights of the bottom line performance. As the streaming service listens to and retrieves new replies of the client's tweets, we use Perspective API to get the toxicity scores for the replies. Types of toxicity include general toxicity, identity attack, insult, profanity, threat, sexual explicity, flirtation; each of the entries is a floating point number. The mean scores of all replies by certain adversary is used as one feature.

During the register process, we also get access to the network information of the adversary, namely how many accounts the adversary follows are also muted or blocked by the client. The normalized count is used as a second feature.

Sentiment and network features with the labels obtained from the initial labeling process are fitted into a binary classification logistic regression model. Test performance is evaluated by humans. Model training details and the results are described in the following section.

CHAPTER 5

MODEL TRAINING AND RESULTS

First, we finished the development and testing for the logistic regression models.

Practical issues include lack of data points for training. This problem is more drastic than last semester as all replies from one replier are aggregated into one single feature vector to represent that replier. We solved this problem by introducing mock data mentioned in Section 3.2.

Here, we provide two modes, test mode and normal workflow mode.

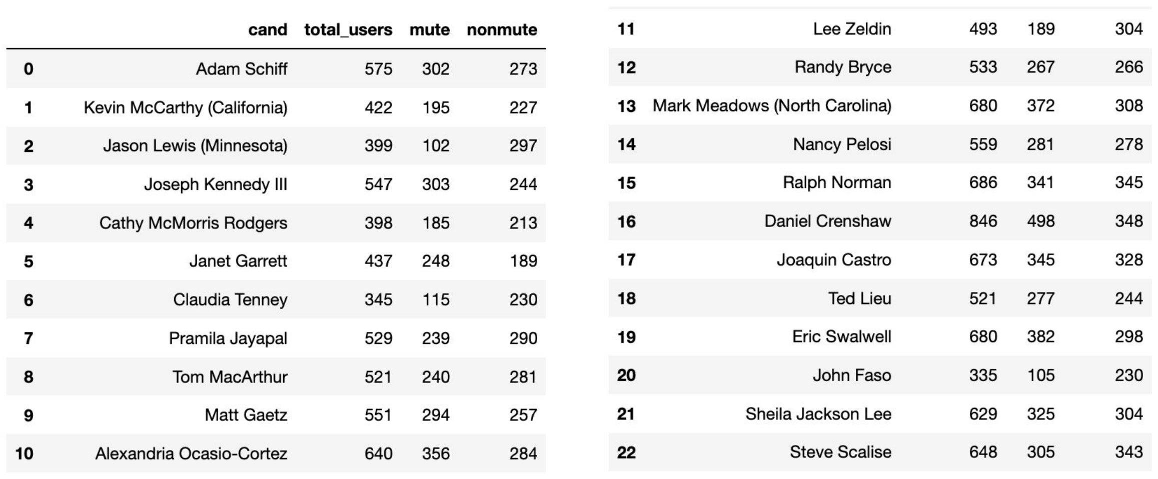
5.1 Test Mode

In the test mode, we pretend that there are several political candidates who have registered to our platform and thus we can mute their reply users according to the results of the model. This mode is mainly used for backend model evaluation.

The data we are using for the mock candidates is retrieved from what we were using last semester. Our work from last semester has already labeled whether the tweet (not users) is toxicity or not by the 'toxicity' value we get from perspective api, if the value is larger than 0.7

then we label the reply tweet as *toxic*, otherwise, we label it as *non-toxic*. Using the labeled tweet reply, then, for each reply user, we set their feature value as the average feature value of each tweet reply we get from the perspective api, and label whether the user is muted (as 1) or unmuted (as 0) according to the majority tweets label (whether the majority replies this reply user posts is toxic or not).

Then, we get the top 23 candidates that have most reply users, and surprisingly, the average value of the muted and unmuted reply users are exactly even (which has shown in the table below).



	cand	total_users	mute	nonmute
0	Adam Schiff	575	302	273
1	Kevin McCarthy (California)	422	195	227
2	Jason Lewis (Minnesota)	399	102	297
3	Joseph Kennedy III	547	303	244
4	Cathy McMorris Rodgers	398	185	213
5	Janet Garrett	437	248	189
6	Claudia Tenney	345	115	230
7	Pramila Jayapal	529	239	290
8	Tom MacArthur	521	240	281
9	Matt Gaetz	551	294	257
10	Alexandria Ocasio-Cortez	640	356	284
11	Lee Zeldin	493	189	304
12	Randy Bryce	533	267	266
13	Mark Meadows (North Carolina)	680	372	308
14	Nancy Pelosi	559	281	278
15	Ralph Norman	686	341	345
16	Daniel Crenshaw	846	498	348
17	Joaquin Castro	673	345	328
18	Ted Lieu	521	277	244
19	Eric Swalwell	680	382	298
20	John Faso	335	105	230
21	Sheila Jackson Lee	629	325	304
22	Steve Scalise	648	305	343

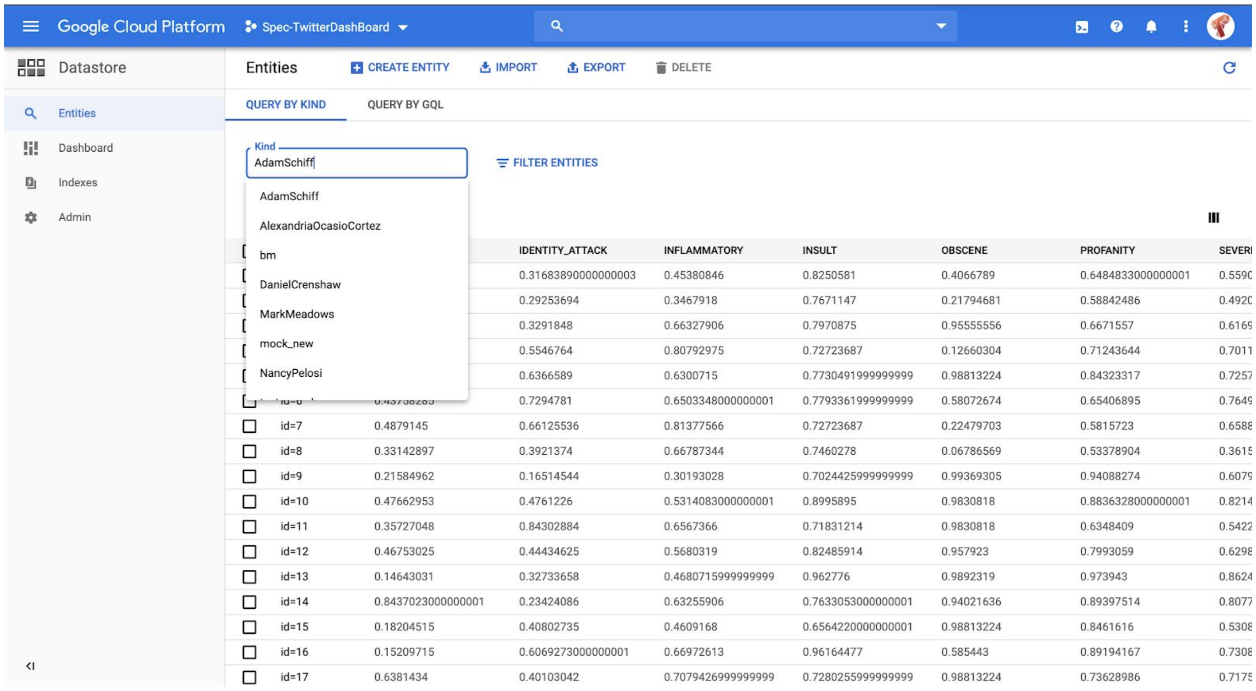
Figure 9: Label Statistics of Top 23 Candidates

Total Reply Users	549.869565
Muted Reply Users	272.434783
Unmuted Reply Users	272.434783

Table 1: Average Label Statistics on Top Candidates

According to the graph, we select Daniel Crenshaw, Alexandria Ocasio-Cortez, Mark Meadows (North Carolina), Adam Schiff and Nancy Pelosi as our mock users, as they have a relatively considerable number of unique reply users. We insert their corresponding data (includes reply users, features get from the Perspective api, and labels we generate from their features) into the datastore, each candidate corresponds to a kind of table, for training by the model in compute engine. So, if political candidates registered into our system, when the corn job for backend model runs, it will generate one model for each user, thus, we can get the model for each user and the new label for new reply users for each candidate.

Here is a screenshot for our datastore in GCP, which contains the five political candidates we select from above.

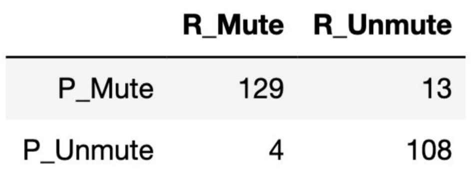


		IDENTITY_ATTACK	INFLAMMATORY	INSULT	OBSCENE	PROFANITY	SEVER
		0.31683890000000003	0.45380846	0.8250581	0.4066789	0.6484833000000001	0.559
		0.29253694	0.3467918	0.7671147	0.21794681	0.58842486	0.492
		0.3291848	0.66327906	0.7970875	0.95555556	0.6671557	0.616
		0.5546764	0.80792975	0.72723687	0.12660304	0.71243644	0.7011
		0.6366589	0.6300715	0.7730491999999999	0.98813224	0.84323317	0.7257
[illegible]	[illegible]	0.7294781	0.6503348000000001	0.7793361999999999	0.58072674	0.65406895	0.764
id=7	0.4879145	0.66125536	0.81377566	0.72723687	0.22479703	0.5815723	0.658
id=8	0.33142897	0.3921374	0.66787344	0.7460278	0.06786569	0.53378904	0.361
id=9	0.21584962	0.16514544	0.30193028	0.7024425999999999	0.99369305	0.94088274	0.607
id=10	0.47662953	0.4761226	0.5314083000000001	0.8995895	0.9830818	0.8836328000000001	0.8214
id=11	0.35727048	0.84302884	0.6567366	0.71831214	0.9830818	0.6348409	0.5422
id=12	0.46753025	0.44434625	0.5680319	0.82485914	0.957923	0.7993059	0.6298
id=13	0.14643031	0.32733658	0.4680715999999999	0.962776	0.9892319	0.973943	0.8624
id=14	0.8437023000000001	0.23424086	0.63255906	0.7633053000000001	0.94021636	0.89397514	0.8077
id=15	0.18204515	0.40802735	0.4609168	0.6564220000000001	0.98813224	0.8461616	0.5308
id=16	0.15209715	0.6069273000000001	0.66972613	0.96164477	0.585443	0.89194167	0.7308
id=17	0.6381434	0.40103042	0.7079426999999999	0.7280255999999999	0.98813224	0.73628986	0.7175

Figure 10: Database Architecture

To evaluate our test, we split the users into train, test sets, going through our logistic regression model on the computer engine. And the confusion matrix for each candidate are below:

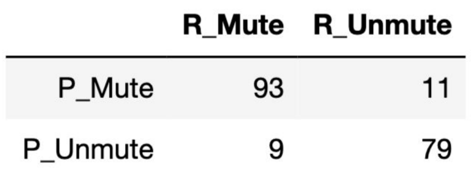
The **accuracy** for Daniel Crenshaw is: 0.933071, and the confusion matrix is:



	R_Mute	R_Unmute
P_Mute	129	13
P_Unmute	4	108

Figure 11: Confusion Matrix of Daniel Crenshaw

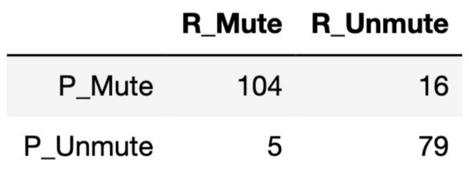
The **accuracy** for Alexandria Ocasio-Cortez is: 0.89583, and the confusion matrix is:



	R_Mute	R_Unmute
P_Mute	93	11
P_Unmute	9	79

Figure 12: Confusion Matrix of Alexandria Ocasio-Cortez

The **accuracy** for Mark Meadows (North Carolina) is: 0.89706, and the confusion matrix is:



	R_Mute	R_Unmute
P_Mute	104	16
P_Unmute	5	79

Figure 13: Confusion Matrix of Mark Meadows

The **accuracy** for Adam Schiff is: 0.8786127, and the confusion matrix is:

	R_Mute	R_Unmute
P_Mute	86	17
P_Unmute	4	66

Figure 14: Confusion Matrix of Adam Schiff

The **accuracy** for Nancy Pelosi is: 0.875, and the confusion matrix is:

	R_Mute	R_Unmute
P_Mute	78	20
P_Unmute	1	69

Figure 15: Confusion Matrix of Nancy Pelosi

And overall, the average accuracy for all the 23 candidates is: 0.90834.

5.2 Normal Workflow Mode

In this mode, we require Twitter users to connect our platform through the normal register process. Their real tweets will be extracted and some complementary replies are inserted into his user table in order to simulate the back end model training process.

The mock data comes from the 1.5 million tweet-reply pairs we collected and cleaned during last semester. As we've already have the toxicity scores for them, we first selected a sample of 4938 pairs from 269 replier accounts. Each selected repliers replied more than 10 tweets, and among the replies it showed a well-balanced composition of toxic and non-toxic contents. Continuing with last semester's heuristics, toxic replies are those with toxicity score higher than 0.7. These selected pairs are inserted into a Data Store table *mock* without the toxicity scores as they are the simulated raw data that we retrieve directly with Twitter API.

When the web service starts, as a new test user comes to register, we used the workflow described in Application Logic part to get real account content. On the other hand, we retrieved from the mock data tweet-reply pairs from 100 different repliers in the background. We passed these records together with the real data retrieved from Twitter API through Perspective API to simulate the real workflow and inserted the record into the user's exclusive table. In this way, when the cron job of training models starts, there will be a considerable amount of data points (repliers) to train the personalized model. The sampling process also makes sure that different test users have a different set of training samples, in order to really personalize the models.

Another perspective of the problem is that we do not have many user accounts that authorized our service for testing. We tested the service with 3 of our own Twitter accounts. In the future, we might need to work on recruiting for test accounts. As for time constraint, we had no luck with the full model. However, we've also set up the running environment for training deep learning networks for future convenience.

CHAPTER 6

FUTURE WORK

There are some future works related to this project, and the following are some possible directions:

1. Other features that could be added to this project

   a. The dashboard page that can display visualizations other than the repliers we muted for our users, such as monthly aggregated statistics of harassment content, most severe type of toxicity for the user, nudges and suggestions on how to manage user's online appearance.

   b. Improve the performance of the machine learning models.

   c. More data is needed to train our models.

   d. Better model architecture. The goal for the model is to mute users for our customers, which is actually a subjective matter. We could follow the methodology used when developing the direction model during last semester and incorporate more heuristic based features that might boost the models' performance.

2. Improve the user interface to offer users more seamless experience integrated to Twitter

3. Incorporate more social media platforms, such as Facebook, Instagram, Reddit, in order to offer our users a more holistic view of their online appearance and become a one-stop shop for online harassment detection.

CHAPTER 7

CONCLUSION

During this semester, we implemented a personalized website which could help Twitter users mute the repliers automatically and display the muted repliers list. We also conduct two user test cases each evaluated by one of the team members, and the feedback is listed as follows. As both test cases are not heavy Twitter users, we used samples from mock data set to supplement training data. At query time, we once again retrieved another sample as the test data and evaluated model performance on this new sample. Evaluation was done by going into the "retrieve user" page for each automatically muted account, and observing the overall toxicity level of the replies from this account. We also kept a copy of the query data set offline in order to check the overall toxicity level of those accounts that were not muted by the model. These two observations show that our model made sense and reached an overall satisfactory precision and recall.

To sum up, we built a basic skeleton for this project, and more effort is needed for better results. We separately implemented the model part and the website part to make it much easier to edit and work on for future development. There is still space for considerable improvement of this project.

REFERENCES

[1] Google Cloud Platform.April 2018. From: https://cloud.google.com/

[2] Hua et al. 2019. Towards Measuring Adversarial Twitter Interactions against Candidates in the US Midterm Elections

[3] Jigsaw et al. 2018. Perspective API. From: https://www.perspectiveapi.com/#/

[4] M. Naaman, Y. Hua, J. Sun, and X. Sun, May 2019. Online Harassment Campaign: Tweets Adversary Direction Detection.

[5] Twitter Api Reference. From
https://developer.twitter.com/en/docs/tweets/search/api-reference

[6] Y. reina. July 2019. Toxic train BERT Base pytorch. From
https://www.kaggle.com/yuval6967/toxic-train-bert-base-pytorch?scriptVersionId=16389235

[7] Thomas Wolf et al. PyTorch Pretrained BERT: The Big & Extending Repository of pretrained Transformers.