Assignment 4

Due **Weds Oct 31st** by 11:59PM to Canvas
(but will be graded in class the following Monday)

For this assignment, you'll create a "chatbot" (really a dialog system) to provide useful information. You'll build your bot on Twitter -- people will write to your bot by starting their messages with @<yourbot> and your bot will figure out what it should say back.

You can choose to implement your chatbot however you want, but this doc will overview a fairly straightforward way in Python. There are a number of "bot" platforms out there now, which (at least in theory) allow you to write your bot once and deploy it to many different bot platforms (e.g., Facebook, Alexa, Google Now, etc.). For instance, you might check out DialogFlow.

This assignment has the following main parts:

- Set Up Twitter Bot
- Find and Prepare Information for Your Bot to Talk About
- Set Up What Your Bot Can Recognize and Say
- Track Interactions with Your Bot
- Test with Users and Iterate
- Grading Overview and Rubric
- Bonus Opportunities

As you complete the instructions below, you'll notice there are some questions (in red/bold). Please answer these in a document.

Set Up Twitter Bot

Your chatbot will work as follows -- you'll run a python script, which will check to see if there are any messages addressed to your bot that have not yet been answered, and, if so, it will figure out how to respond. This script will be run periodically to check for new messages. For the purposes of this assignment, you can just run your script manually.

You could just type to your chatbot from the commandline, but that would be pretty boring. So, we're going to make a Twitter bot. This introduces some interesting challenges, and is something that real people actually write chatbots for! In fact, networks of Twitter bots can apparently throw elections, ruin whole world orders, etc. Be careful with your new powers.

The first step is to sign up on Twitter for API access. This has all gotten a bit more complicated as Twitter has over the years attempted to deal with various threats. Nevertheless, despite all the additional questions, Jeff did this in < 5 minutes.

Apply for a developer account here:
https://apps.twitter.com

As you go through the process of signing up for a developer account, reflect on the questions you are asked, why you are being asked them, and how you think they will serve the intended purpose (or not).

If you get stuck somewhere, or are otherwise unable to sign up for a developer account. Please message “project-discuss” on Slack, so we know. You can still do the rest of the assignment (don’t get stuck!), but you’ll of course need to modify it a bit so you’re not using your own account. For instance, have your bot just print its replies to the commandline instead of tweeting them back.

We’ll be using a Python library called “Tweepy” to interact with the Twitter API. You can install that in your python environment using pip:

```
pip install tweepy
```

To authenticate Tweepy to use your Twitter account, you’ll need to provide it with your Twitter developer credentials.

```
consumer_key = 'XXX'
consumer_secret = 'XXX'
access_token = 'XXX'
access_token_secret = 'XXX'
```

You can obtain these from your Twitter developer page. Create a new application, go to the Keys section, and copy/paste and/or generate them from there.

This all assumes that you don’t mind signing up and doing all of this under your main Twitter account. You are more than welcome to sign up for another Twitter account, and following these instructions there. You can even sign up as a developer on one account, and then access another account’s feed (that’s why there are two sets of tokens). But, for these instructions, we’re assuming you’re doing everything on one account.

I recommend putting these in a separate python file, e.g., `credentials.py`, and then importing that file into your main script. That helps separate out your secret information, and will make it easier for you to, for instance, turn in your bot code without also giving Jeff and Joseph access to your Twitter account.

```
import credentials
```

Once you do this, you can test out whether or not your script has access to Twitter with the following:

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)

user = api.me()
print (user.name)
```

This snippet of code should access your Twitter account, fetch your name stored there, and then print it out. If it doesn't, go back and debug!

The next step is to find a tweet that mentioned your bot, which are the messages that your bot will respond to. Here's some template code for searching for tweets with Tweepy:

```
####
# Define the search
#####
query = '@jeffbigham'
max_tweets = 100

####
# Do the search
#####
searched_tweets = []
last_id = -1
while len(searched_tweets) < max_tweets:
    count = max_tweets - len(searched_tweets)
    try:
        new_tweets = api.search(q=query, count=count,
max_id=str(last_id - 1))
        if not new_tweets:
            break
        searched_tweets.extend(new_tweets)
        last_id = new_tweets[-1].id
    except tweepy.TweepError as e:
        # depending on TweepError.code, one may want to retry or wait
        # to keep things simple, we will give up on an error
        break

####
# Iterate over the search
```

```
#####
for status in searched_tweets:
  # do something with all these tweets
  print    (status)
```

If you run this snippet, it will print out the messy JSON version of all the statuses returned. You can see what sorts of things each of the `status` objects contain. Some important ones are:

`text` -- the text of the returned tweet
`author.screen_name` - the Twitter username of the user who sent the tweet
`id` - id of the tweet, which you can use to reply to it, or search only for tweets that were posted after it

And, of course, you want to be able to send new tweets to, which is done like this:
`api.update_status('I\'m on a boat!')`

You will likely want to not just send a status, but actually reply to the original tweet. You can combine some of what you've learned here to do that:

```
api.update_status(
  'this is a reply! @' + status.author.screen_name,
  status.id_str
)
```

You have to include the @-mention in the tweet, or it won't show up as a reply.

Of course, this is just the tip of the iceberg with what you can do with Tweepy. Fortunately, there is a lot of documentation online. You might start with the official Tweepy docs.

Find and Prepare Information for Your Bot to Talk About

Now that you can do basic replies with your bot, it's time to make it do something useful!

You are free to choose any topic for you bot, subject to the requirement that your bot be able to provide access to some database of information. Our suggested database for your bot to operate over is the course registration information for next semester (Spring 2019).

All of the information for courses is located here:
https://enr-apps.as.cmu.edu/assets/SOC/sched_layout_spring.htm

You'll notice that this i just a file containing information, and so your first step is to parse this

information into a form that you can access with python. There are a lot of ways to do this, and you can do it however you want. You'll notice that the page is presented in very ugly, but also very regular, HTML. You could attempt to use a python HTML Parser to help you. Probably that is not going to be as simple as using a sequence of text processing steps, including regular expressions, given how regular this HTML is, but again that is up to you :)

Regular expressions allow you to formally define a pattern that the computer can then match against an input string. In this case, you might define a pattern that captures all of the columns in the data on the page. Below is an example in python. You might also want to consult the python regular expression documentation. As a side note, Jeff is old, and would find it much easier to do this in an amazing programming language called Perl that the younger generation has forgotten.

But, here's a python example:

```
>>> import re
>>>
>>> inputstring = '<tr><td>Yo</td><td>What\'s up</td></tr>'
>>> pattern = '<tr><td>(.*?)</td><td>(.*?)</td></tr>'
>>>
>>> match = re.search(pattern, inputstring)
>>>
>>> match.group(0) # whole string
"<tr><td>Yo</td><td>What's up</td></tr>"
>>> match.group(1) # first group
'Yo'
>>> match.group(2) # second group
"What's up"
>>>
```

We would advise against trying to set up a real "database" since it's probably a lot of work and overkill for the size of the data that you're dealing with here. One option is to simply keep your data in a nicely formatted file. For instance, you might turn this file into a JSON representation that can be written to a file by your parsing script, and then read in by your bot script.

Here is python JSON encoding and decoding documentation.

It's fine and encouraged to develop this script iteratively. One tip is to start with a smaller subset of the whole dataset.

Set Up What Your Bot Can Recognize and Say

Now that you have your dataset and have parsed the information that it contains, the next step is to figure out what your bot can actually say. We'll call these "intents", and you'll represent them first by imagining what phrases your users will use to invoke them. For instance, let's imagine a weather bot… in that case, user's might say things like:

```
What's the weather today?
Will it rain tomorrow?
What's the day out of the next 4 in which it is least likely to rain?
What's the temperature right now?
How hot is it out?
How cold is it out?
Hey bot, tell me the weather.
… … …
```

If you look at these, some of the phrases are asking for different things. But some seem to be asking for the same thing, *i.e.*, "What's the weather today?" and "Hey bot, tell me the weather". Phrases that are meant to convey basically the same thing are said to represent the same "intent."

Entities in this context are ways that your intents can be parameterized. For instance, saying "What's the weather?" only makes sense if you know where the user is located, which we might not know. It probably makes more sense for users to ask things like, "What is the weather in Pittsburgh?" In this case, "Pittsburgh" is a Location entity.

Thus far the examples have all been "single turn" examples. That is, the user says something, and then the bot responds based on the information contained in that single message. This is how most of the so-called "conversational assistants" work today. But, real conversation is uses multiple turns between partners to establish shared information and context to aid future communication.

A simple example might be to respond differently based on what information you have, and what information you need. If your bot already knew where the user was somehow, then a query like, "What is the weather?" can be answered immediately. If you don't know where the user is, then the bot might need to ask, *e.g.*, "Where are you right now?"

For this project, you'll come up with at least 5 different intents that you want to support for users of your bot. You should come up with 2-5 different ways that people may ask these things of your bot. At least 2 of these should have the potential for a follow-up questions if the bot doesn't currently have all of the information that it needs to answer the question. At least 3 of the intents

should require some sort of entity to work (*e.g.,* a time, location, academic major, etc.).

In the base form of your chatbot, you'll just detect exact phrases (or templates). You can use regular expressions for this as well. A simple, but surprisingly powerful, approach to making your bot more general is to match on keywords instead of exact phrases. For instance, if the user includes the word "weather", your chatbot might always choose to return the current weather. This is called "keyword spotting", and is the key to a lot of early dialog systems. Using this approach, it doesn't matter what the user says before or after, as long as they say "weather" they'll hear something about the weather.

The origin of the "keyword spotting" approach is from AT&T, which was trying to build an automatic triage system for incoming phone calls. As the story goes, people would call and give the robot some long story about the context and backstory and their frustrations, but would often mention some keywords in their long speech that would be informative. For instance, they might somewhere along the way say "bill" or "cancel", and detecting these keywords were sufficient for directing the call to the right department.

You should adapt at least one of your intents to a keyword spotting approach. For our fictional weather bot, we could just look for messages that include the word "weather" and then return the current weather forecast for the user's current location.

Track Interactions with Your Bot

Chatbots that don't remember prior interactions with the user can be very frustrating. It would be annoying if every time you asked a bot for the weather if it asked you (again) where you are. For this part of the assignment, you'll record basic information that you learn about the user that is relevant. You'll also store enough about past interactions so you avoid doing basic dumb things like responding more than once to the same query.

You could track these interactions with a database, but we're going to keep this very simple and just use files. You can do this however you want, but in our implementation we have a file for the overall chatbot, and a separate file for each user that the chatbot interacts with.

For example, if you have stored the information that you want into an object, you can export it to a file in JSON format using the following:

```
import json

with open('data.json', 'w') as outfile:
  json.dump(data, outfile)
```

You can then read in this JSON into an object, using the following:

```
import json

with open('data.json') as f:
  data = json.load(f)
```

You will have to decide what you want to store, but some ideas for things that might be useful to store include the id of the last tweet that was retrieved, information that you might have learned from the user, and other "state" relevant to the dialog your bot is having with the user.

What did you choose to store and why? What limitations were there in what you were able to store and manage effectively?

Test with Users and Iterate

A really hard challenge with creating chatbots is informing what they know how to recognize. People will naturally express similar kinds of queries in different ways. Sometimes the ways they will express those things will differ in small ways, and you might be able to reason around those differences, as we did with the "keyword spotting" approach. Other times it won't be as easy, or you'll find it is too easy to get confused.

In this part, you'll ask three participants to interact with your bot. You'll give the user high-level information about what the domain of the bot is, and then see how they interact with it. Ask each of the participants to ask your chatbot at least three different things. Record how they interact with your bot.

How did what your participants ask compare to the intents that you chose to implement? How did participants react when the chatbot didn't respond correctly, or couldn't interpret what they asked for?

After each participate study, update your bot to attempt to address how that participant interacted with your chatbot. This is a 3 cycle iteration.

Did any of the changes you made during earlier iterations allow the chatbot to better respond to later participants? How many participants do you think you'd need to try the chatbot with before it would be very likely to respond correctly to a new participant?

Grading Overview and Rubric

Please turn your assignment in on Canvas -- Please turn in a .zip file of the code for your chatbot and dataset parsing script, and a PDF of your assignment write-up (including answers to the questions in your assignment).

[1pt] Twitter bot is able to observe new mentions, and respond
[2pt] You prepared a dataset (possibly the registrar dataset) for your bot to be able to respond to -- you have written a script to parse it, and put it into some structured format (likely JSON)
[2pt] Twitter bot is able to recognize at least 5 different intents, and each is expressed in multiple ways.
[2pt] Twitter bot is able to detect at least 2 entities, and use them to parameterize responses.
[2pt] Twitter bot can have at least 2 different kinds of multi-turn conversations.
[2pt] Twitter bot tracks and stores some information about the user to inform future interactions.
[2pt] You tested your Twitter bot with at least 3 users, and updated your Twitter bot after each iteration.
[4pt] Answers to the questions asked in this assignment (bold and in red)

Bonus Opportunities

[+2pt] Adapt your chatbot to work on a different domain (on a different dataset)
[+2pt] Adapt your chatbot to work on a different platform (Facebook, Alexa, Google Now, etc.)
[+2pt] Have another 5 participants interact with your bot, iterate on your chatbot based on their responses, and discuss how this further iteration impacted the bot's ability to interact with each additional participant.