Collecting Tweets

Using Search to collect tweets

Outline

- HCDE user module Search.py
  - Instantiation
  - Parameters
- Collecting, storing with pickle
- Collecting, storing in a db

Using Search.py – command line

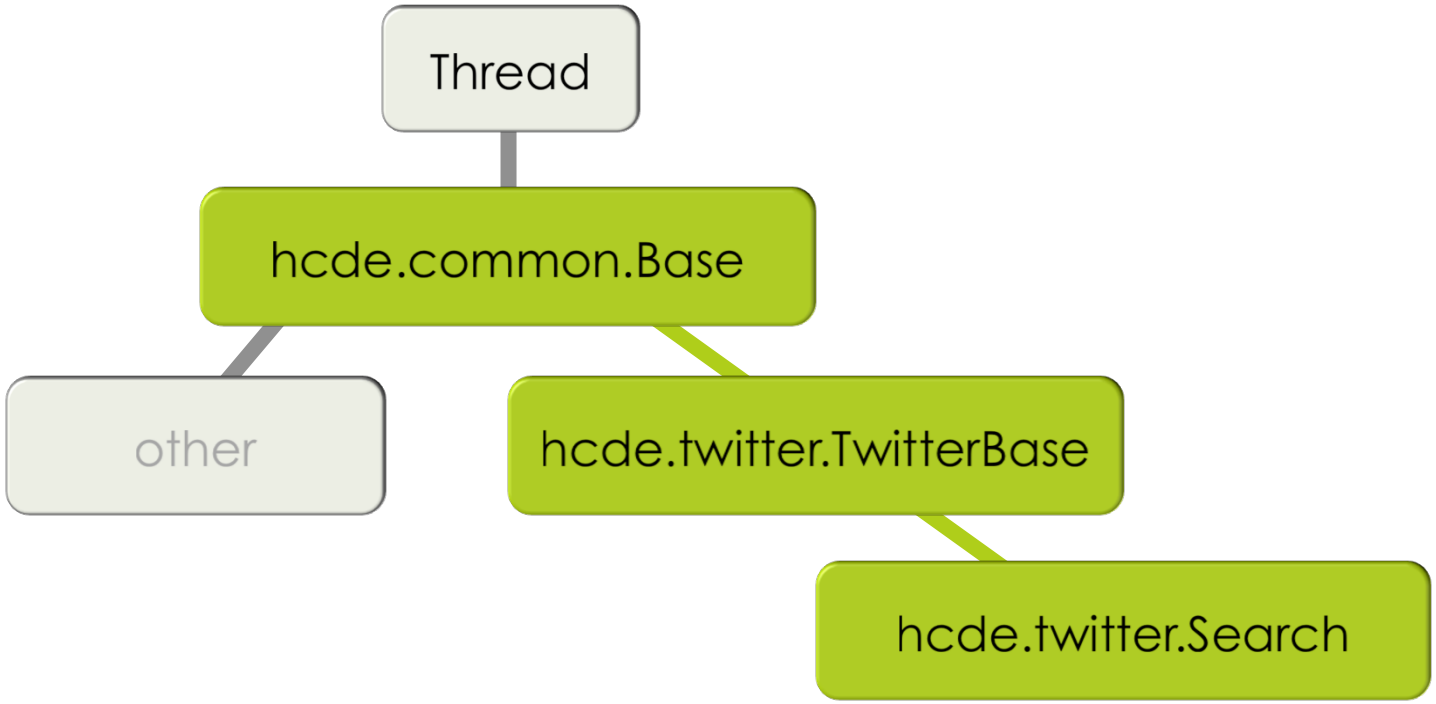
- Saw a quick example of Search.py in prior lecture
  - "Intro to Twitter & HCDE Module Setup"
- Remember where this is in hcde user module …
  - hcde/twitter
- Can get a little help – run without command line parameters
  - `python Search.py`

Using Search.py – quick demo

- Quick demo of search from the command line

Reviewing Search.py code

- Search.py object hierarchy



Base.py

- hcde.common.Base
  - Code for common things that help make web API requests
    - get/set http header values
      - User-Agent
    - manage a response message queue
    - minimal http error handling
    - throttling
    - know something about authenticated requests
    - basic make_request() function
    - thread start, terminate

TwitterBase.py

- hcde.twitter.TwitterBase
  - Extend and elaborate hcde.common.Base
    - Support Twitter specific things
    - Rate limiting on requests
    - Twitter continuation requests
    - Twitter http errors

Search.py

- hcde.twitter.Search
  - Extend and elaborate hcde.twitter.TwitterBase
    - Support Search specific things
    - Which URL for making requests
    - What search parameters to support
    - Handle search specific continuations

Reading the Search() code

- Reading structure
  - Import statements
  - Object definition
    - Object specific data
    - Object methods
  - Command line parameter parsing
  - main() procedure
    - An example of how to use the object

Reading the Search() code

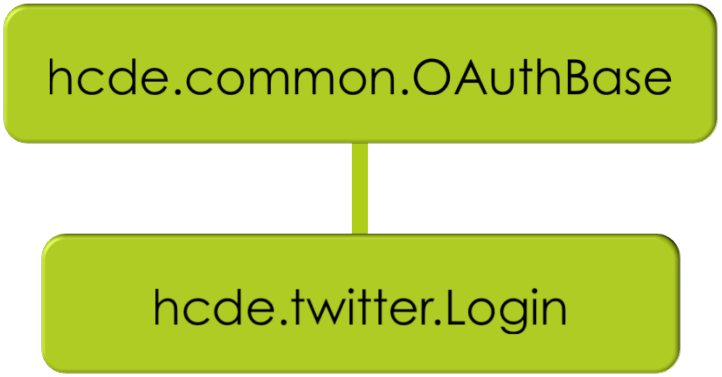
- Demo/Review source code

Reading the Search() code

- Demo/Review source code
- Small problem, we didn't cover all that Login stuff

Reviewing Login.py code

- Login.py object hierarchy



OAuthBase.py

- hcde.common.OAuthBase
  - Implement encoding and making a request
  - User secret/key pair management
    - file/directory storage
    - request/retrieval

Login.py

- hcde.twitter.Login
  - Twitter specific aspects of OAuth
  - URLs for requesting tokens

  - Will retrieve and save user secret/key pair

Search() object

- Now that we know something about how the Search() object and the Login() object work, we can put them together to to make a request.

Demo (setup oauth login)

```
import sys, json
from hcde.twitter.Login import Login
from hcde.twitter.Search import Search
from hcde.twitter.auth_settings import *
# First create an authorized login object
app = "HCDE530Test01"
user = "dwmcphd" # needs to be your twitter user
app_keys = TWITTER_APP_OAUTH_PAIR(app=app)
app_token_fname = TWITTER_APP_TOKEN_FNAME(app=app)
lg = Login(app_name=app, app_user=user, token_fname=app_token_fname)
lg.set_consumer_key(consumer_key=app_keys['consumer_key'])
lg.set_consumer_secret(consumer_secret=app_keys['consumer_secret'])
lg.login()
```

Demo (setup search)

```
# Assuming imports and lg (Login object from prior page)
search = Search()
search.set_auth_obj(obj=lg) # all Twitter requests require auth
search.set_user_agent(agent="ie")
search.set_throttling(True)
search.set_query_result_type(rt="recent")
search.set_page_size(sz=100)
terms = "seahawks"
search.set_query_terms(terms)
# NOW WHAT?
# A little tricky here ... If you just want to make a single request just
search.make_request()
# easy, no threading, just a request
```

Demo (getting search result)

```
# Assuming Search() object and search from prior page
# WHERE IS THE RESULT?
# Check for messages
print search.messages()
# Get the message from the message queue
response = search.get_message()
# The response is a list of dictionary items, json response was converted
print len(response)
#
for t in response:
          print json.dumps(t,indent=4,sort_keys=True)
          print type(t)
```

Brief Review

- To this point …
  - Understand the object hierarchy
  - Know where to look for Search() and Login()
  - Know how to create objects
  - Know how to make a request
- Need to be able to store tweets
  - Fortunately, we already saw an example of using pickle
  - We're saved!

Need a pickle routine

```
import pickle
def pickle_data(fname="",d={}):
    if fname:
        fout = open(fname, "w")
        pickle.dump(d,fout)
        fout.close()
    return
#
# Now we can save the response
pickle_data("data1.pickle",response)
```

Need a loop to save many tweets

```
import time
def collection_loop(search=None, lots=10, fname="data"):
    count = 1
    total = 0
    while(count <= lots):
        print "Making request %d, got"%(count),
        search.make_request()
        response = search.get_message()
        if( response ):
            print "%d tweets"%(len(response))
            total = total + len(response)
            data_fname = fname+"%04d.pickle"%(count)
            pickle_data(data_fname,response)
        else:
            break
        time.sleep(5.0)
        count += 1
    return total
```

Sample loop

```
# Now we can collect lots of tweets
# assuming we have a Search object called search
total = collection_loop(lots=5, fname="tweet_data", search=search)
Making request 1, got 100 tweets
Making request 2, got 100 tweets
Making request 3, got 100 tweets
Making request 4, got 100 tweets
Making request 5, got 100 tweets
print total
500
```

Collecting in DB

- Started working with DBs last week
  - Query by date
- Recall the example DB and schema
  - hcde.data.db.example
- Some sample code that works with DB
  - hcde.data.example

collect_tweets.py

- Basic collector
  - Hook basic Search() object to a DB
  - Saves tweets and some user data

- Can be run from the command line

```
python collect_tweets.py –auth HCDE530Test01 -user dwmcphd -page_size 100 -cont -query #seahawks
```