Computer Memory and Web Scraping

Matthew Seguin

Preliminary Importing of Packages

```
import os
import time
import inspect
try:
      import numpy as np
except ModuleNotFoundError:
      with open('requirements.txt') as f:
            lines = f.readlines()
            install_numpy_command = lines[1].strip("\n")
            os.system(install_numpy_command)
try:
      import pandas as pd
except ModuleNotFoundError:
      with open('requirements.txt') as f:
            lines = f.readlines()
            install_pandas_command = lines[2].strip("\n")
            os.system(install_pandas_command)
```

1.

Computer memory is used to access smaller bits of information that are often used for short, temporary processes. Memory can not store information permanently, whatever data it was storing gets erased once it loses power. Hard drives and solid state drives (or disks) are used to permanently store information and often times have much more capacity to store large amounts of information. The drawback is that disks are much slower than memory. So your computer often uses memory to store information you are repeatedly using in your current session while it has power but uses the disk to store information that needs to be accessed on future sessions.

All information used came from my own personal experience building computers and from the provided notes at:

https://36-750.github.io/tools/computer-architecture/

2.

a.

We are given that a numeric value generally takes up 8 bytes of storage, we will assume that for our purposes this is the case for all numbers we use. So if we have a matrix with each row being 20 columns we have $20 \times 8 = 160$ bytes of data for each row. We want to take up about 16 MB (megabytes, a post on Ed confirmed Mb was a typo) total. There are $1,000,000$ bytes in a megabyte. So we want to have $16 \times 1,000,000 = 16,000,000$ bytes of data which would take $\frac{16,000,000}{160} = 100,000$ rows of data.

The process I am using here generates a random vector of values from a standard normal distribution in the form of a list data type each time there is an entry in list [1, 2, 3, ..., 100000] which is just 100000 times. This is a list of lists but we want it as a matrix (or array) so I used np.array which is a numpy command to convert it to an array. At the end I print the dimensions (row x column) and amount of memory the array takes up.

```
x = [np.random.normal(loc = 0,
                      scale = 1,
                      size = 20
                      ) for _ in range(100000)
    ]
x = np.array(x)
byte_size = x.nbytes
print("Dimensions of matrix: ",
      x.shape
      )
print("Number of bytes (B): ",
      byte_size
      )
print("Number of megabytes (MB): ",
      byte_size/1000000
      )
```

```
Dimensions of matrix:  (100000, 20)
Number of bytes (B):  16000000
Number of megabytes (MB):  16.0
```

b.

```
x = x.round(decimals = 12)

pd.DataFrame(x).to_csv('x.csv',
                       header = False,
                       index = False
                       )
print(f"{str(os.path.getsize('x.csv')/1e6)} MB")

pd.DataFrame(x).to_pickle('x.pkl',
                          compression = None
                          )
print(f"{str(os.path.getsize('x.pkl')/1e6)} MB")
```

```
30.877551 MB
16.000572 MB
```

- Explaining the size of the csv file:

  We know that for a number x drawn from a standard normal distribution $\mathbb{P}[x > 0] = \mathbb{P}[x < 0] = \frac{1}{2}$ and that it is exceedingly unlikely that $|x| \geq 10$ so we can say that each entry will have 13 numeric digits (since we round to 12 decimal points). Then each of these will have a decimal point which is another character, and about half of them will have a $-$ in front since about half will be negative. This means that on average each data point has $13+1+0.5 = 14.5$ characters in it. We have a total of $20 \times 100,000 = 2,000,000$ data points which gives $2,000,000 \times 14.5 = 29,000,000$ characters from our data. However, a csv implicitly adds a comma between each data point in a row. So for each row since we have 20 data points we will have $20 - 1 = 19$ commas which makes $19 \times 100,000 = 1,900,000$ total commas. So our total number of characters is $29,000,000 + 1,900,000 = 30,900,000$. Now a csv file stores data as text which uses 1 byte (B) per character so this gives us an estimated size of $30,900,000$ bytes or $\frac{30,900,000}{1,000,000} = 30.9$ megabytes (MB), very close to the true value.

- Explaining the size of the pkl file:

  We know that a pkl file stores data as binary and that numeric values take 8 bytes (B) each. We have a total of $20 \times 100,000 = 2,000,000$ data points so this will take about $2,000,000 \times 8 = 16,000,000$ bytes (B) or equivalently $\frac{16,000,000}{1,000,000} = 16$ megabytes (MB) to store.

- Considering how this would look if we used 4 decimal places instead:

  Let us consider how the size of the files would change if we instead use 4 decimal places. Each data point would have 5 numeric characters, one decimal point, and a $-$ in half of our numbers giving an average of $5 + 1 + 0.5 = 6.5$ text characters per data point for our csv file. This means we have a total of about $6.5 \times 2,000,000 = 13,000,000$ characters stored from our data. The number of commas remains the same giving us a total of $13,000,000 + 1,900,000 = 14,900,000$ text characters each taking one byte (B). So we have an expected size of $14,900,000$ bytes (B) or $\frac{9,000,000}{1,000,000} = 9$ megabytes (MB) of storage for the csv file.

  However, the pkl file still uses 8 bytes per data point it just has a large number of 0s at the end it still counts, and so the size stays unchanged. So we would expect the csv to be more efficient in terms of storage space in this case.

Now let us test the performance of using 4 decimal places (we can use the same code as before replacing 12 with 4 and changing file names).

```
x = x.round(decimals = 4)

pd.DataFrame(x).to_csv('x_alt.csv',
                       header = False,
                       index = False
                       )
print(f"{str(os.path.getsize('x_alt.csv')/1e6)} MB")

pd.DataFrame(x).to_pickle('x_alt.pkl',
                          compression = None
                          )
print(f"{str(os.path.getsize('x_alt.pkl')/1e6)} MB")
```

```
14.878835 MB
16.000572 MB
```

In this case we actually saved space when using the csv file, matching the theory we had before.

c.

Here we will quickly test saving our data to a csv all in one column. What I do is I use flatten on our list of lists x so that it becomes a single list with all the values from the original data in it. Then I make it a pandas data frame and write a csv file. Then we check the size just as before. Note: This is done after we rounded all of the data points to 4 decimal places so we compare the size to that of the csv with values of 4 decimal places.

```
pd.DataFrame(x.flatten()).to_csv("x_one_col.csv",
             header = False,
             index = False
             )
print(f"{str(os.path.getsize('x_one_col.csv')/1e6)} MB")
```

```
16.778835 MB
```

We can see the size is more or less the same. Since there are no more commas we should expect to see a decrease in file size of $1,900,000$ bytes (B) or 1.9 megabytes (MB) since this is how much the commas were taking before but we don't see this. This is because in place of a comma between each element in a row we now have a newline character after each row in place of the comma. They both take up 1 byte so there is no change in file size.

d.

We will read both the csv and pkl file noting the time it takes for each using the time package.

```
t_0 = time.time()
pd.read_csv("x.csv")
t_1 = time.time()
pd.read_pickle("x.pkl")
t_2 = time.time()
print("Time to read the csv file: ",
      t_1 - t_0,
      "(s)"
      )
print("Time to read the pkl file: ",
      t_2 - t_1,
      "(s)"
      )
```

```
Time to read the csv file:  0.1676480770111084 (s)
Time to read the pkl file:  0.0030121803283691406 (s)
```

We can see that it takes much more time to read the csv file (over 10 times the amount).

e.

Here we will read only the first 10,000 rows of the csv, timing it.

```
t_0 = time.time()
pd.read_csv("x.csv",
            nrows = 10000
            )
t_1 = time.time()
print("Time to read first 10,000 rows of the csv file: ",
      t_1 - t_0,
      "(s)"
      )
```

```
Time to read first 10,000 rows of the csv file:  0.01919722557067871 (s)
```

f.

Here we will test reading a chunk from the start of the csv and another chunk of the same size starting some amount in already, timing it.

```
t_0 = time.time()
pd.read_csv("x.csv",
            nrows = 10000
            )
t_1 = time.time()
pd.read_csv("x.csv",
            nrows = 10000,
            skiprows = 10000
            )
t_2 = time.time()
pd.read_csv("x.csv",
            nrows = 10000,
            skiprows = 30000
            )
t_3 = time.time()
pd.read_csv("x.csv",
            nrows = 10000,
            skiprows = 50000
            )
t_4 = time.time()
pd.read_csv("x.csv",
            nrows = 10000,
            skiprows = 70000
            )
t_5 = time.time()
print("Time to read first 10,000 rows of the csv file: ",
      t_1 - t_0,
      "(s)"
      )
print("Time to read 10,000 rows of the csv file after skipping 10,000 rows: ",
      t_2 - t_1,
      "(s)"
      )
print("Time to read 10,000 rows of the csv file after skipping 30,000 rows: ",
      t_3 - t_2,
      "(s)"
      )
print("Time to read 10,000 rows of the csv file after skipping 50,000 rows: ",
      t_4 - t_3,
      "(s)"
      )
print("Time to read 10,000 rows of the csv file after skipping 70,000 rows: ",
      t_5 - t_4,
      "(s)"
      )
```

```
Time to read first 10,000 rows of the csv file:  0.018573999404907227 (s)
```

```
Time to read 10,000 rows of the csv file after skipping 10,000 rows:  0.027431488037109375 (s)
Time to read 10,000 rows of the csv file after skipping 30,000 rows:  0.03832578659057617 (s)
Time to read 10,000 rows of the csv file after skipping 50,000 rows:  0.05379652976989746 (s)
Time to read 10,000 rows of the csv file after skipping 70,000 rows:  0.06696581840515137 (s)
```

As you can see the time to read 10,000 rows only increases as the number of rows we skip increases. This makes sense as python needs to count more rows while it is skipping. So it doesn't make to read the file in chunks as opposed to reading it all at once as the time to read later chunks only increases whereas if you read it all at once no time is spent counting rows to skip, you just read one chunk until you have reached the end and start reading the next. Python could theoretically read in chunks then combine at the end but this is not time efficient as shown above so it likely doesn't read in chunks and skip over data.

3.

I am using separate .py files in order to lint for problem 4. I added comments, used blank lines and whitespace, used informative variable names, wrote a function description for each function, and added try except functionality for installing needed packages. A couple things that I changed after linting include using "is" instead of "==" when comparing the type of an object and changing a bare except to "except ModuleNotFoundError" for fixing the automatic package installation.

4.

a.

First let us get our soup data, note that here we want to manually download the html file from the search as Google Scholar does not formally allow webscraping. Once we have manually downloaded the html we can read it into python. Function documentation is attached

```
from HTMLReader import HTMLReader as htmlreader
import HTMLReader
print(inspect.getsource(HTMLReader))

htmlpath = "Michael Jordan - Google Scholar.html"
MichaelJordanData = htmlreader(htmlpath)
```

```
def HTMLReader(PATH):
    '''
    Used to read in an html file and
    return a parseable BeautifulSoup object
    '''

    # Import packages, bs4 is used to help parse through the html
    try:
        from bs4 import BeautifulSoup as bs
    except ModuleNotFoundError:
        import os
        with open('requirements.txt') as f:
            lines = f.readlines()
            install_bs4_command = lines[1].strip("\n")
            os.system(install_bs4_command)

    # Open the file with latin-1 encoding
    # (I was getting errors when using UTF-8)
    html_file = open(PATH, 'r', encoding='latin-1')

    #Read the data and assign to variable
    html = html_file.read()

    # Turn raw html into BeautifulSoup object for parsing
    soup = bs(html, 'html.parser')

    # Return BeautifulSoup object
    return(soup)
```

I have created a general function which should find the Scholar ID from Google Scholar for a recognized Scholar. It uses several tags common in html such as “a” for hyperlink and “table” for a table as well as what I believe to be Google Scholar specific things like the Scholar ID appearing after “user=” in a link.

```
from FindScholarID import FindScholarID as IDFinder
import FindScholarID
print(inspect.getsource(FindScholarID))
```

```
Element = IDFinder(MichaelJordanData, returnelement = True)
UserID = IDFinder(MichaelJordanData)
```

```
def FindScholarID(html, returnelement = False):
    '''
    Used to find the Scholar ID or optionally
    the whole element containing it in an
    html file which can be provided as a path
    or a BeautifulSoup python object and
    return either the element with the
    Scholar ID or just the Scholar ID
    '''

    # Check to see if a filepath or an html object was
    # given and import html if needed
    if type(html) is str:
        from HTMLReader import HTMLReader as htmlreader
        html = htmlreader(html)

    # Find all of the tables in the html
    tables = html.body.find_all("table")

    # Find all of the hyperlinks inside
    # each table (a is an html hyperlink tag)
    links = [table.a for table in tables]

    # Get the elements from links as strings
    elements = [str(link) for link in links]

    # Search for which element has "user="
    element = elements["user=" in elements]

    # If we want to return the whole element return
    if returnelement:
        return(element)

    # Otherwise we want just the ID so find
    # where the query field starts for it
    start_index = element.find("user=")

    # Define a function that finds a given
    # query field given a string and an
    # index to start at
    def LocateField(string, start):
        '''
        Used to find a query field from a link
        given a starting point from the link
        and the link as a string
        '''

        # Create start and end point variables
```

```
        begin = ""
        end = ""

        # Initialize iterators
        i, j = start, start

        # While we haven't found both
        # values keep iterating
        while begin == "" or end == "":
            # If the current value is the signal
            # of the assignment of a query field
            # and we haven't already assigned a
            # begin point then assign begin
            if string[i] == "=" and begin == "":
                begin = i + 1
            # Otherwise move one character forward
            # Note: we are moving forward because
            # Using find on a string gives the
            # location of the first entry of what
            # We are looking for so the "=" is later
            else:
                i += 1
            # If the current value is the signal
            # of the start of another query
            # field and we haven't already
            # assigned an end point then assign end
            if string[j] == "&" and end == "":
                end = j
            # Otherwise move one character forward
            else:
                j += 1
        # Return the string from the starting
        # to the end point found
        return(string[begin:end])

    # Use our locate function to find the user
    # query field which is the Scholar ID
    UserID = LocateField(element, start_index)

    # Return the Scholar ID
    return(UserID)
```

```
print(Element)
print(UserID)
```

```
<a href="https://scholar.google.com/citations?user=yxUduqMAAAAJ&amp;hl=en&amp;oi=ao"><b>Michael </b>I. <b>Jordan</b></a>
yxUduqMAAAAJ
```

b.

First we can see the structure of the link is “https://scholar.google.com/citations?user=UserID&hl=en&oi=ao” we want to be able to change the Scholar ID which comes from “user=” and construct our link. We saw from

above that only one User ID was found and that it corresponds to Michael Jordan so we will use that directly when constructing our link. The function works with other scholars as well. Note: we are not running the function because we do not want to webscrape.

```
MichaelJordanID = str(UserID)

from GoogleScholarScraper import GoogleScholarScraper as scraper
import GoogleScholarScraper
print(inspect.getsource(GoogleScholarScraper))

# Commented out because we don't want to run if we already have the html
# scholar_profile = scraper(MichaelJordanID)
```

```
def GoogleScholarScraper(UserID):
    '''
    Used to create the Google Scholar
    Profile link given a Scholar ID
    then download the html page and
    read it into python, returning
    it as a BeautifulSoup object
    '''

    # Import subprocess package so we can make
    # query and import HTMLReader to read in
    # the downloaded html
    import subprocess
    from HTMLReader import HTMLReader as htmlreader

    # Deconstruct sections of link based on
    # looking at some example links
    base_link = "https://scholar.google.com"
    query_citations = "/citations?user="
    ending_filters = "&amp;hl=en&amp;oi=ao"

    # Construct link based on inputted UserID
    Scholar_Profile_Link = (base_link +
                            query_citations +
                            UserID +
                            ending_filters)

    # Download html
    htmlpath = "scholar_profile.html"
    subprocess.run(["curl", "-o " + htmlpath,
                    Scholar_Profile_Link],
                    capture_output=True)

    # Import to python using our previous function
    soup = htmlreader(htmlpath)

    # Return BeautifulSoup object
    return(soup)
```

c.

Here we use several tags common in html such as “a” for hyperlink, “table” for a table, “tr” for a table row, and “td” which I believe to be something like table div, as well as what I believe to be Google Scholar specific attributes like the citations summary table having “id=gsc_a_t”, the fact that all of: title, authors, and journal information are kept in the first column of the table together. Similar things are noted in the function documentation.

```
from PandasCreator import PandasCreator as pd_create
import PandasCreator
print(inspect.getsource(PandasCreator))

try:
      CitationsTable = pd_create(scholar_profile)
except NameError:
      CitationsTable = pd_create("scholar_profile.html")
```

```
def PandasCreator(html):
    '''
    Used to create a pandas dataframe with
    rows for each publication from a scholar
    and a column for the title, authors,
    journal information, year of publication,
    and number of citations from an html
    file which can be provided as a path
    or a BeautifulSoup python object and
    return that pandas dataframe
    '''

    # Import pandas to make dataframe object at the end
    try:
      import pandas as pd
    except ModuleNotFoundError:
      import os
      with open('requirements.txt') as f:
            lines = f.readlines()
            install_pandas_command = lines[2].strip("\n")
            os.system(install_pandas_command)

    # Check to see if a filepath or an html object was
    # given and import html if needed
    if type(html) is str:
        from HTMLReader import HTMLReader as htmlreader
        html = htmlreader(html)

    # Find the table corresponding to the publications.
    #
    # Note: From inspecting the webpage we can see
    # Google Scholar seems to use the gsc_a_t tag for this
    #
    # Since we know there will be only one matching table
```

```
# we can use this to find the table and extract the
# data from the list
table = html.body.find_all("table", attrs = {"id": "gsc_a_t"})[0]

# Make a list of all the rows from this table
# Note: tr is table row
rows = table.find_all("tr")

# Make a matrix that has the columns from the table
# by searching for all td (which I think is table div)
elem_matrix = [row.find_all("td") for row in rows]

# Remove all empty objects
# Note: the title of the table has no td, making it empty here
elem_matrix = [elem for elem in elem_matrix if elem != []]

# Get the publication details from each row
# which is in the first element of the row
# Note: this has the title, authors, and journal all combined
publications = [row[0].contents for row in elem_matrix]

# Get the title from each publication
# which is in the first element of the
# publication contents
titles = [pub[0].contents[0] for pub in publications]

# Get the author from each publication
# which is in the second element of the
# publication contents
authors = [pub[1].contents[0] for pub in publications]

# Get the journal information from each publication
# which is in the third element of the
# publication contents
journals = [pub[2].contents[0] for pub in publications]

# Get the number of citations from each row
# which is in the second element of the row
num_citations = [row[1].a.contents[0] for row in elem_matrix]

# Get the year published from each row
# which is in the third element of the row
year_published = [row[2].span.contents[0] for row in elem_matrix]

# Create pandas data frame
pd_table = pd.DataFrame(
    {"Title": titles,
     "Authors": authors,
     "Journal Information": journals,
     "Year Published": year_published,
     "Citation Count": num_citations
     }
```

```
                                )

    # Return our pandas data frame
    return(pd_table)
```

```
CitationsTable
```

	Title	Authors	Journal Information	Year Published	Citation Count
0	Latent dirichlet allocation	DM Blei, AY Ng, MI Jordan	Journal of machine Learning research 3 (Jan), ...	2003	54461
1	On spectral clustering: Analysis and an algorithm	A Ng, M Jordan, Y Weiss	Advances in neural information processing syst...	2001	12435
2	Machine learning: Trends, perspectives, and pr...	MI Jordan, TM Mitchell	Science 349 (6245), 255-260	2015	9424
3	Trust Region Policy Optimization	J Schulman	arXiv preprint arXiv:1502.05477	2015	8466
4	Adaptive mixtures of local experts	RA Jacobs, MI Jordan, SJ Nowlan, GE Hinton	Neural computation 3 (1), 79-87	1991	6103
5	Learning transferable features with deep adapt...	M Long, Y Cao, J Wang, M Jordan	International conference on machine learning, ...	2015	5938
6	Graphical models, exponential families, and va...	MJ Wainwright, MI Jordan	Foundations and Trends® in Machine Learning 1 ...	2008	5530
7	An introduction to variational methods for gra...	MI Jordan, Z Ghahramani, TS Jaakkola, LK Saul	Machine learning 37, 183-233	1999	5434
8	Sharing clusters among related groups: Hierarc...	Y Teh, M Jordan, M Beal, D Blei	Advances in neural information processing syst...	2004	5420
9	An internal model for sensorimotor integration	DM Wolpert, Z Ghahramani, MI Jordan	Science 269 (5232), 1880-1882	1995	4238
10	Hierarchical mixtures of experts and the EM al...	MI Jordan, RA Jacobs	Neural computation 6 (2), 181-214	1994	4195
11	Distance metric learning with application to c...	E Xing, M Jordan, SJ Russell, A Ng	Advances in neural information processing syst...	2002	4033
12	High-dimensional continuous control using gene...	J Schulman, P Moritz, S Levine, M Jordan, P Ab...	arXiv preprint arXiv:1506.02438	2015	3819
13	On discriminative vs. generative classifiers: ...	A Ng, MI Jordan	Advances in Neural Information Processing Syst...	2002	3667
14	An introduction to MCMC for machine learning	C Andrieu, N De Freitas, A Doucet, MI Jordan	Machine learning 50, 5-43	2003	3596
15	Optimal feedback control as a theory of motor ...	E Todorov, MI Jordan	Nature neuroscience 5 (11), 1226-1235	2002	3565
16	Learning the kernel matrix with semidefinite p...	GRG Lanckriet, N Cristianini, P Bartlett, LE G...	Journal of Machine learning research 5 (Jan), ...	2004	3140
17	Kalman filtering with intermittent observations	B Sinopoli, L Schenato, M Franceschetti, K Poo...	IEEE transactions on Automatic Control 49 (9),...	2004	2986
18	Deep transfer learning with joint adaptation n...	M Long, H Zhu, J Wang, MI Jordan	International conference on machine learning, ...	2017	2867
19	Theoretically principled trade-off between rob...	H Zhang, Y Yu, J Jiao, E Xing, L El Ghaoui, M ...	International conference on machine learning, ...	2019	2630

d.

In this file so far I have used the packages os, time, inspect, numpy (as np), pandas (as pd), bs4 (inside my HTMLReader file), and subprocess (inside my GoogleScholarScraper file) so I want to make sure the user has these installed with pip (out of these the only things that need installing are numpy, pandas, and bs4). I added a try/except portion to where I am importing these libraries so that everything should automatically install.

e.

You will have seen that I used a function to find the Scholar ID earlier, I have a function that should work to find the ID provided an BeautifulSoup html object in python or the path to an html file. Here is the documentation again. Note: I added an option to this function that allows me to return the entire element containing the ID as well as that was asked in problem 4.a. as well.

```
print(inspect.getsource(FindScholarID))
```

```
def FindScholarID(html, returnelement = False):
    '''
    Used to find the Scholar ID or optionally
    the whole element containing it in an
    html file which can be provided as a path
    or a BeautifulSoup python object and
    return either the element with the
    Scholar ID or just the Scholar ID
    '''

    # Check to see if a filepath or an html object was
    # given and import html if needed
    if type(html) is str:
        from HTMLReader import HTMLReader as htmlreader
        html = htmlreader(html)

    # Find all of the tables in the html
```

```
tables = html.body.find_all("table")

# Find all of the hyperlinks inside
# each table (a is an html hyperlink tag)
links = [table.a for table in tables]

# Get the elements from links as strings
elements = [str(link) for link in links]

# Search for which element has "user="
element = elements["user=" in elements]

# If we want to return the whole element return
if returnelement:
    return(element)

# Otherwise we want just the ID so find
# where the query field starts for it
start_index = element.find("user=")

# Define a function that finds a given
# query field given a string and an
# index to start at
def LocateField(string, start):
    '''
    Used to find a query field from a link
    given a starting point from the link
    and the link as a string
    '''

    # Create start and end point variables
    begin = ""
    end = ""

    # Initialize iterators
    i, j = start, start

    # While we haven't found both
    # values keep iterating
    while begin == "" or end == "":
        # If the current value is the signal
        # of the assignment of a query field
        # and we haven't already assigned a
        # begin point then assign begin
        if string[i] == "=" and begin == "":
            begin = i + 1
        # Otherwise move one character forward
        # Note: we are moving forward because
        # Using find on a string gives the
        # location of the first entry of what
        # We are looking for so the "=" is later
        else:
```

```
                i += 1
            # If the current value is the signal
            # of the start of another query
            # field and we haven't already
            # assigned an end point then assign end
            if string[j] == "&" and end == "":
                end = j
            # Otherwise move one character forward
            else:
                j += 1
        # Return the string from the starting
        # to the end point found
        return(string[begin:end])

    # Use our locate function to find the user
    # query field which is the Scholar ID
    UserID = LocateField(element, start_index)

    # Return the Scholar ID
    return(UserID)
```

5.

Here is what robots.txt says

```
with open('robots.txt') as f:
            for line in f:
                  print(line)
```

```
User-agent: *

Disallow: /search

Disallow: /index.html

Disallow: /scholar

Disallow: /citations?

Allow: /citations?user=

Disallow: /citations?*cstart=

Disallow: /citations?user=*%40

Disallow: /citations?user=*@

Allow: /citations?view_op=list_classic_articles

Allow: /citations?view_op=metrics_intro
```

```
Allow: /citations?view_op=new_profile

Allow: /citations?view_op=sitemap

Allow: /citations?view_op=top_venues



User-agent: Twitterbot

Disallow:



User-agent: facebookexternalhit

Disallow:



User-agent: PetalBot

Disallow: /
```

We can see that searching is disallowed as well as finding citations (except if we already know the user) as well as many other things, so it seems to be against Google Scholar's terms of service to webscrape (in general).