CIS 191: Linux and Unix Spring 2020

Homework 0: Getting yourself a Linux

This assignment is due ***by noon*** **on Monday** , February 3, 2020.

Part of the reason Linux has gotten so popular is because it is so easy to get your hands on! Between Penn's resources and the internet, you can have your pick of linux environments.

- Eniac is SEAS's main server, and runs openSUSE. Every SEAS student has an account on Eniac (instructions in Part II).
- SEAS has a number of computer labs with linux workstations. Your home directory will be the same as your home directory on Eniac.
- *Many* linux distributions (distros) are freely available online, and you can download them and install them on your own computer. You can either:
  - replace your current operating system (drastic!); or
  - install linux side-by-side with your own OS (less drastic but prone to data loss, so **back up your data!!**); or
  - run linux in a virtual machine. (safe! Instructions in Part I.)

Part 0: Enroll in Piazza

Sign up for Piazza at piazza.com/upenn/spring2020/cis191 .

Part I: Install Linux on a Virtual Machine (Optional)

- Download VirtualBox, a virtual machine software program.
- Download an iso image of your preferred distro (lightweight distros are recommended).
- Use VirtualBox to create a new virtual machine using the iso image you downloaded.
- For better performance and usability of your guest operating system, it is recommended that you install VirtualBox Guest Additions (go here to find out about the features of guest additions). If you're running a debian-based VM such as Ubuntu, installing guest additions may be as simple as running the command:

  ```
  sudo apt install virtualbox-guest-utils virtualbox-guest-dkms virtualbox-guest-x11
  ```

  Alternatively you can try the following. Click on Devices in VirtualBox Mananger, click on Insert Guest Additions CD image, open a terminal in your virtual machine, navigate to the /media directory to where the Guest Additions image has been mounted, and install like so:

  ```
  cd /media/[your username]/Vbox_GAs_[version]
  sudo ./VBoxLinuxAdditions.run
  ```

- Once you have the guest additions installed, you will have mouse pointer integration, shared folders, better video support, a shared clipboard, and other useful features. To set up a shared folder between your host and virtual machine, go to the settings for your virtual machine in VirtualBox, look for the shared folders option, and add the folder to be shared with whichever options you desire. If you do not choose the Read-Only option, then you can edit your shared folder from both your machine and the virtual machine. If you do not choose a mount-point, then the shared folder will be mounted in the /media directory with the prefix "sf_".

Part II: Log Into Eniac

Eniac (the main server for the engineering school) will be the main platform we use in this class.

NOTE : If you are not a SEAS student, you will need to create a SEAS account .

Open up a terminal window, and enter the following into the command prompt:

```
$ ssh username @eniac.seas.upenn.edu
```

where *username* is your username. Enter your SEAS/pennkey password when prompted. (We will explain the `ssh` command in a later lecture).

Part III: Submission

Open your terminal and run the following command:

```
$ uname -a > mylinuxdistro.txt
```

This creates a text file recording the details of the linux distribution you are running. (Check it out using `less` or `cat` !) To submit this file you must do two things.
First, you need to copy your `mylinuxdisto.txt` file from your local computer to Eniac, using the `scp` command, which works like `ssh` but copies files.

```
$ scp mylinuxdistro.txt PENNKEY@eniac.seas.upenn.edu:~
```

This will put a copy of `mylinuxdistro.txt` in your home directory (`~`) on Eniac; if you prefer another directory (e.g. `~/cis191`) then replace `~` with `~/cis191` .
Next, you need to **ssh into Eniac** and use the `turnin` command to turn in your file. Make sure you are in the same directory as `mylinuxdistro.txt` on Eniac.

```
$ turnin -c cis191 -p hw0 mylinuxdistro.txt
```

You can verify that your file was submitted successfully with the following:

```
$ turnin -c cis191 -p hw0 -v
```

In future assignments, you will be using the `turnin` command in the following (more general) way:

```
$ turnin -c cis191 -p NAME_OF_ASSIGNMENT FILE1 FILE2 ...FILEn
```

CIS 191: Linux and Unix Spring 2020

Homework 1

This assignment is due ***noon*** **on Monday**, February 10, 2020.

Part I: Setting up your environment

For this assignment you should download the file maze.tar. The file extension `tar` means that this file is a *tape archive*, a single file that contains a whole collection of files.

The `maze` directory is quite large, and you will find it is too cumbersome to navigate by brute force. Therefore you will be using your command line tools to explore the maze in the next section. Remember to use google and `man` pages to figure out what commands to use and how to use them!

Part II: Exploring the maze

To complete the assignment, you will need to record answers in `instructions.txt`. In that file, you will find a number of instructions and questions regarding the contents of the `maze` directory. Fill out the answers in `instructions.txt` and perform the changes to `maze` as requested.

Part III.1: Make SSHing a Whole Lot Easier

Are you getting tired of typing out a whole ssh command for every single Eniac login? No more! You can create an ssh config file to quickly perform secure logins to even the most strangely named servers.

From your home directory, `cd` into `.ssh/`. It's a hidden folder, notated by the . in front of its name, which can only be seen with `ls -a`. If you don't already have a config file here, `touch config` to create one, and use your favorite text editor to edit it.

The format for an ssh config file is incredibly simple. For each server that you'll want to ssh onto, all you need are three lines of code. The format is as follows (fill in the italics with real information):

```
Host server1
    HostName something.stuff
    User my_name

Host server2
    HostName whatever.thing
    User my_name
```

`config` Entry	**Function**
`Host`	Think of this like an alias. When you want to ssh onto this server, you'll simply type the name you specify here. `eniac` would be a great `Host` for your new Eniac config entry.
`HostName`	The actual address of the server; that is, the part after the `@`.
`User`	Your username

For this assignment, create an ssh config file that allows you to log into your Eniac account by simply commanding `ssh eniac`. **Make a copy of your config file, rename it to `PENNKEY_config`, and save it for later.**

Part III.2: Make SSHing a Whole Lot Easier

Are you getting tired of typing out your whole password for every Eniac login? No more! You can use RSA encryption to make it so that Eniac

will automatically verify your identity without you having to enter a password.

First, you must create a pair of authentication keys. Your *private key* will be kept secret, only available locally on your machine. Your *public key* can be distributed to anybody. If you don't already have a private/public key pair on your local machine, create one by running the command, which will generate the following output:

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/USERNAME/.ssh/id_rsa):
Created directory '/home/USERNAME/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/USERNAME/.ssh/id_rsa.
Your public key has been saved in /home/USERNAME/.ssh/id_rsa.pub.
The key fingerprint is:
3e:4f:05:79:3a:9f:96:7c:3b:ad:e9:58:37:bc:37:e4 USERNAME@MACHINE
```

The defaults are good, although you may wish to create a passphrase which you will need to enter if you ever want to change you private key.

Next, you need to create the `.ssh` on your eniac account if it does not already exist. Finally, you need to append your public key for your local machine (see the result of `ssh-keygen`) to the file `~/.ssh/authorized_keys` on your eniac account. To do this, come up with a command that has the form

```
$ somecommand | ssh eniac 'someothercommand'
```

Record the command you used in a file `response.txt`.

After this is done, when you ssh onto eniac you should not have to enter a password.

Part IV: Streaming Music over the network

In this part, you will use ssh to stream a music file off of eniac and play it on your local VM. First, we need to install software to play the music file. The package we will be using is called mplayer. To install packages on Ubuntu, you can use the command `apt-get install` followed by the name of the package to install. Note: you may get a message about needing to be root. Good thing we just covered this in lecture!

In your host machine (Windows or OSX), make sure your volume is unmuted and is turned up decently high (at least 75%). Turn the volume up to about 50% inside your VM.

The song that you will play is in a file named mus.mp3 in the home directory of the cis191 account (which is aptly named cis191). Remember that in addition to just opening a shell, ssh can be used in a pipeline to execute a single command and then exit. What we want to do in this case is connect to eniac, print the file to our local machine over the ssh tunnel, and direct the output of this tunnel into our music player. Using this form of ssh, cat the previously mentioned music file on eniac and pipe the result to "mplayer -". That is, the command you run should be of the form

```
ssh eniac 'somecommand' | mplayer -
```

(The trailing hyphen is important). If everything is right, the music should start playing.

Copy the command you used to play the music into the `response.txt` file.

Part V: Setup and Configure an Apache Web Server

For this part of the assignment, you will install a web server (apache) and change some ***very*** basic parameters. This may seem intimidating at first, but it's actually quite simple:

1. First, install apache2 using `apt-get install` on your virtual machine.
2. After apache2 finishes installing, direct your web browser to `localhost` or `127.0.0.1`. You should see a plain webpage saying "It worked!". This means apache2 was installed correctly and is now running.

   "BUT WAIT A MINUTE! DOES THIS JUST MEAN I RAN ONE COMMAND AND ALL OF A SUDDEN I CAN MAKE WEBSITES THAT THE WHOLE WORLD CAN SEE?"
   Yes it does! (At least if you have certain network settings.)
3. By default, apache2 looks in the directory `/var/www` for files that make up the website. You will need to change the configuration files so that apache looks at the `/home/yourusername/www` directory instead.

4. In your home directory, make a new directory called `www`. Inside of that directory, create a new file called `index.html`. If you know some HTML, feel free to add some here, otherwise, use `wget` to grab a basic html template from here and customize it at least a little bit. Note: make sure you give read permission to the `other` category for `index.html`.
5. To change where apache2 looks for website files, you must edit the `/etc/apache2/sites-available/000-default.conf` and `/etc/apache2/apache2.conf` files. Every instance where the path `/var/www` is mentioned needs to be changed to `/home/yourusername/www` in each of the files.
6. After you are done editing, save and close the file. You will have to restart apache2 in order for your changes to take effect. To do this, run the command

   ```
   sudo service apache2 restart
   ```

7. Once apache2 has restarted, direct your browser to `localhost` or `127.0.0.1` again. It should now display the text in `/home/yourhomedirectory/www/index.html`.

You don't need to turn anything in for this part, but you will use the html files you created for Part VI.

Part VI: Host your homepage on Eniac

Did you know you can use Engineering school's server to host your homepage? This service is not only available to faculty and course accounts, but to anyone who has a seas account as well. Follow these steps to put the html file you just created up on Eniac (if you have not done so yet).

First log into your eniac account as usual and you should see a directory called `html` in your home directory. If not, just create an empty one and give it read and execute permissions to users in the group and other categories (i.e. chmod 755 html).

Next use `scp` from your virtual machine to copy the html files you created in Part V to the `html` directory.

Make sure you give read permission to the `other` category for `index.html`.

Point your browser to `www.seas.upenn.edu/~PENNKEY` to check it out. You don't need to turn in anything for this part, but **we will visit your webpage and grade on completeness.**

Submission

1. Take your edited `instructions.txt` file, the edited `maze` directory and any additional files you may have created (e.g. `output.txt`) from part II as well as your `PENNKEY_config` and `response.txt` file from parts III.2 and IV, and add them to a `tar` file named `hw1.tar`.
2. ssh into eniac and submit your tar file using the command

   ```
   $ turnin -c cis191 -p hw1 hw1.tar
   ```

3. You can verify that your file was submitted successfully by running

   ```
   $ turnin -c cis191 -p hw1 -v
   ```

CIS 191: Linux and Unix Spring 2020

Homework 2

Due: noon on Monday, March 2, 2020.

NOTE : For this assignment use the gawk implementation of awk.

Part I: A Quick Intro to arrays in awk

What does the following awk program do?

```
#!/usr/bin/gawk -f
!A[$0] {
  print
  A[$0]=1
}
```

Record your answer in a file named response.txt (Hint: Experiment on lines that have a single word in them).

Part II: Exploring the /etc/passwd file using awk

Traditionally, the /etc/passwd file is used to keep track of every registered user that has access to a system. The /etc/passwd file is a colon-separated file that contains the following information:

- User name
- Encrypted password
- User ID number (UID)
- User's group ID number (GID)
- Full name of the user (GECOS)
- User home directory
- Login shell

Here is an example of an entry in the /etc/passwd file:

```
solomon:x:1000:1001:solomon:/home/solomon:/bin/bash
```

Take a look at the /etc/passwd file on eniac to familiarize yourself with the format.

The permissions for /etc/passwd are by default set so that it is world readable, that is, so that it can be read by any user on the system.

The password field originally contained an encrypted login password. However, for security reasons, the encrypted passwords are now contained on another file, /etc/shadow, that cannot be read by ordinary users. This field now merely contains the letter x to indicate that a password has been assigned to the user and is required for authentication. If this field is empty, the user can log in without a password.

Write an awk script file called names.awk that prints the total number of occurrences of the first name in each full username (i.e. the fifth field of each line in the /etc/passwd file) in order of their frequency, ignoring case. For example, given the following /etc/passwd file:

```
solomonaduol:x:1000:1001:solomon aduol:/home/solomonaduol:/bin/bash
johndoe:x:1002:1003:john doe:/home/johndoe:/bin/bash
solomonmaina:x:1004:1005:SOLOMON maina:/home/solomonmaina:/bin/bash
johnwick:x:1006:1007:john wick:/home/johnwick:/bin/bash
jamesbond:x:1006:1007:james bond:/home/jamesbond:/bin/bash
solomonakasha:x:10010:10011:Solomon akasha:/home/solomonakasha:/bin/bash
```

the command

```
./names.awk /etc/passwd
```

should output

```
solomon-3
john-2
james-1
```

Take a look this page to learn about some of gawk's built-in string and array functions; they can save you some time and keystrokes.

Part III: Simple Encryption with sed

Part III (a)

The message "kjfgrcxko" has been encoded using encrypt.sed, which has the following contents:

```
#!/usr/bin/sed -f
y/trupaovsglimwyxbfekczqjndh/abcdefghijklmnopqrstuvwxyz/
```

Write a sed script called decrypt_using_y.sed that uses the y command to decrypt the message. The following command should output the word "linux":

```
echo linux | ./encrypt.sed | ./decrypt_using_y.sed
```

What do you get when you run the command?

```
echo kjfgrcxko | ./decrypt_using_y.sed
```

Record your response in the file response.txt.

Part III (b)

Write a sed script file called decrypt_using_s.sed that is equivalent to the file decrypt_using_y.sed but only uses the s command. You will probably need to have multiple s commands, one on each line of the sed file. Also, you may assume that the plaintext (i.e. the text before encryption) and the ciphertext (i.e. the encrypted plaintext) will only contain lowercase letters.

Part IV: Learn Vim or Emacs

For this part of the assignment, we will ask you to learn how to use either vim or emacs. We won't expect you to know both, but you should become comfortable using one of them for text editing. Start this part early, since both of them have a steep initial learning curve!

- Learning Emacs: First run `apt install emacs` if you don't already have it installed on your local computer. Emacs comes with a tutorial you can use by typing `Ctrl-h t` while emacs is running. You can also use this guide or any other emacs tutorial on the web.
- Learning Vim: First run `apt install vim` if you don't already have it installed on your machine. Vim comes with a tutorial you can use by entering `vimtutor` at the command prompt. You can also use this guide or any other vim tutorial (check out Vim Adventures for a fun adventure game that teaches you some of the basics!)

You don't have to submit anything for this part of the homework, but it will make your life a lot easier in the rest of this class!

Part V: Trick Out Your Editor

Customizing your editor allows you to have very personal tweaks that suit your coding style and needs. For this next part, download either this if a Vim user, or this if you're rolling with Emacs. **You only need to do one!** Go through the dozen or so uncommented tweaks, and comment on the file what each tweak does. Consider adding a couple of these to your own .vimrc/.emacs file, as these are some nice basic ones to get started with.

When you're done annotating, please **rename the file to either `PENNKEY.vimrc` or `PENNKEY.emacs` .**

Submission

1. Take your `PENNKEY.vimrc`/`PENNKEY.emacs` file, your `response.txt` file, your `names.awk` file, your `decrypt_using_y.sed` file, and your `decrypt_using_s.sed` file and add them to a `tar` file named `hw2.tar` .
2. ssh into eniac and submit your tar file using the command

   ```
   $ turnin -c cis191 -p hw2 hw2.tar
   ```

3. You can verify that your file was submitted successfully by running

   ```
   $ turnin -c cis191 -p hw2 -v
   ```

CIS 191: Linux and Unix Spring 2020

Homework 3

> Due: Noon on Monday, March 16, 2020.

Part A : Managing Assignments and Submissions

Introduction

For this part of assignment, you will be writing a framework out of shell scripts that manages your assignments and submissions for CIS191. You will be writing a toplevel script `cis191.sh` that takes in some command line options to help create, specify, and submit assignments.

Although all options will be accessed through the single top-level script `cis191.sh` called with different options described below, you may wish to use various helper scripts or helper functions to to modularize your code.

> Please comment your code! I will be reading your shell scripts, which means they must be legible! Points may be taken off for excessively messy code.

Part I : Create new assignment

The command

`cis191.sh create` *`hw1`*

should create a new directory *`hw1`* in the toplevel directory if it does not already exist.

Part II : Specifying files to submit

The command

`cis191.sh files` *`hw1 file1 file2 ...`*

specifies the files to be submitted for the assignment *`hw1`*. In particular, the command should record this info in a file *`hw1`*`/.submit`

If this option is run more than once, it should overwrite the previous contents of `.submit`.

Part III : Making backups

As anyone who has had their computer crash knows, backups are important. Your work on eniac gets backed up automatically (see here) so for this assignment, you are going to back up your files by copying them to eniac. The command

`cis191.sh backup` *`hw1`*

should copy the files specified in *`hw1`*`/.submit` to a directory `~/cis191/`*`hw1`* on eniac. The script may need to create this directory if it doesn't already exist. Since eniac makes backups anyway, it is okay to overwrite files from previous backups.

When run with no arguments, the command

`cis191.sh backup`

should back up the files for every assignment.

Part IV : Making periodic backups

Please create a (local) file `mycrontab` that includes a crontab entry in the style discussed in the lecture notes that backs up all of your

assignments (using `cis191.sh backup`) every Tuesday at 1:30pm.

Feel free to add this to your local crontab (run `crontab -e` to access it) if you wish, but for the assignment we only require the standalone file.

Part V : Submitting files

The command

`cis191.sh submit` *`hw1`*

should submit the files specified in *`hw1`*`/.submit` via ssh to eniac; it should copy the files to eniac via your backup procedure and turn them in via `turnin`. Recall that to turn in files for an assignment, you need to run the following command on eniac:

```
$ turnin -c cis191 -p NAME_OF_ASSIGNMENT FILE1 FILE2 ... FILEn
```

Hint: To submit all the files in a directory, remember globbing!

> When testing your code, please do not submit to already existing assignments. Instead, please use assignments `test1`, `test2`, and `test3`, which you can submit things to but I will not grade.

If any of the following files do not exist, you should output a sensible error message and exit:

- the *`hw1`* directory
- the file *`hw1`*`/.submit`
- any of the files specified by *`hw1`*`/.submit`

Overview

option	usage	effect
create	`cis191.sh create` *`assignment-name`*	creates a new directory *assignment-name* that stores the data for that assignment
files	`cis191.sh files` *`assignment-name file1 file2 ...`*	Records the names of files to be submitted the assignment *assignment-name* in a hidden file `assignment-name/.submit`.
backup	`cis191.sh backup` [*`assignment-name`*]	Copies the files specified in `assignment-name/.submit` to eniac. If no assignment name is given, backup up every assignment.
submit	`cis191.sh submit` *`assignment-name`*	Submits the files listed in `assignment-name/.submit` using `turnin` on eniac.

Extra Credit

To get full extra credit points, the command

`cis191.sh due-date` *`hw1 thedate`*

should create a hidden file *`hw1`*`/.due` that contains the value of the due date for *`hw1`*.

You can use the `date` command to use and parse standardized representations of dates and times. For example, `date +\%s` gives the Unix time, the number of seconds since January 1, 1970. Using Unix time you can easily compare two different dates.

If this option is run more than once, it should overwrite the previous contents of `.due`.

When you execute `cis191.sh submit`, compare the due date in *hw1*`/.due` (if it exists) to the current date. If the due date occurred *n* days before the current date, output "You have used *n* late days" in the script.

Part B : The Web Scraper

Write a web scraper in a file named `scraper.sh` that takes the url of a web page, extract all emails and phone numbers from the html of the page, and store these information in organized files.

> For the following assignment, please make use of the extended grep syntax, and not other syntax like perl.

Your scraper should support the followings:

- accepts an argument: a URL, e.g. www.seas.upenn.edu
- Download the page: use `wget` to download the website's content.
- Scrape email/phone numbers: use `grep` to scan the html for anything that looks like an email address or a phone number.
- Store: store phone numbers to phonenumbers.txt, and store email addresses to emails.txt, both in a common format.

Formats

For phone numbers, you should match at least these formats:

- XXX-XXX-XXXX
- (XXX) XXX-XXXX
- (XXX)XXX-XXXX

Feel free to match more, but be careful that you shouldn't match non-phone numbers (like XXXXXXXXXX might just be an integer on the page, not a phone number).

When you store phone numbers in `numbers.txt`, you should pick single format and convert all other phone numbers to that format. *Hint:* You may wish to extract all phone numbers using `grep`, and then use `sed` to convert them all to a common format.

For email addresses, you should at least match strings of the form `name@domain.tld` where `tld` is a top-level domain. For this assignment, it is enough to assume that the `tld` is made up entirely of letters, while the name and domain may consist of letters, numbers, and periods.

In the output file, please store the email addresses in the fom `name (at) domain (dot) tld`.

Hints

Make sure that you use the correct flags for `grep` - by default, it will output an entire line that contains the specified regex. We only want the phone number (Hint hint, `man`).

There are many resources online to help with developing good regular expressions. One good example is rubular.com This site is primarily for developing regular expressions for the ruby programming language, but the regular expression syntax is similar enough between bash and ruby that it is useful here as well. **If you use this tool, you must convert your regular expressions back to Extended Regular Expressions before using them with grep!**

You may use the following site to practice with: http://www.seas.upenn.edu/~cis191/hw/hw6-files/info.txt

scraper.sh enhanced: make your scraper recognize flags

Next, you will add functionality by looping over the arguments to `scraper.sh` and processing flags that will specify extra options. For `scraper.sh`, you will allow the user to optionally specify an input file instead of a web URL, and you will allow the user to specify that they are only looking for phone numbers or only email addresses. Here are some specific commands and what they should do:

- `./scraper.sh -h` - prints a message describing how to use your tool. **Be specific** which specific formats you accept and what format your output files are in.

- `./scraper.sh URL` - default behavior: download the website, then place found numbers in phonenumbers.txt and emails in emails.txt
- `./scraper.sh -f FILENAME` - don't download a website, but instead check the passed in file and place found numbers in phonenumbers.txt and emails in emails.txt
- `./scraper.sh -e URL` - download the website, then place found emails in emails.txt. Do not bother with phone numbers.
- `./scraper.sh -p URL` - download the website, then place found phone numbers in phonenumbers.txt. Do not bother with emails
- `./scraper.sh -e -p -f FILENAME` - same as `./scraper.sh -f FILENAME`
- `./scraper.sh URL -e` - same as `./scraper.sh -e URL`

Don't worry about combining flags (like `-pf FILE`), and assume that the user won't give incorrect input (like `./scraper URL -f FILENAME` or `-f` without a file name immediately after). However, you should expect the `-e -p -f` flags to come in any order.

Extra Credit

Here are some proposed tasks for extra credit (you can come up with other tasks too, but make a private post on Piazza before you implement them):

- Grab other useful information from websites in addition to phone numbers and emails. Modify scraper.sh to support this when called with the appropriate flags.
- So you have successfully written a scraper for a single page; how about going a little further in your evil path and write a crawler+scraper? A crawler starts crawling from a seed page, grab all the urls in this page and recursively crawls all of them as well. Crawling is an interesting topic in itself and a full crawler needs to parse different kinds of URLs, support multi-threading, handle loops, parse `robots.txt`, write to database and much more. You don't need to do any of these (it takes Google thousands of experienced engineers to write their crawler) but anything simple that can continuously scrape from the starting seed page counts. **However, for this assignment, your crawler should halt at some point, and not just crawl links forever.** For example, you may allow the user to specify the depth of links to crawl, or somehow keep track of seen domains.

If you do any extra credit tasks, **make sure to include usage info** in the output of `scraper.sh -h`.

Submission

For this assignment, please submit your `mycrontab` file, your `cis191.sh` script, your `scraper.sh` script, and any other helper scripts you might have. Submit these files on eniac using `turnin`. Note: the following sequence of commands should be sufficient to turn in your code:

```
$ ./cis191.sh create hw3
$ cp cis191.sh mycrontab scraper.sh hw3/ # also any helper scripts you have
$ ./cis191.sh files hw3 cis191.sh mycrontab scraper.sh
$ ./cis191.sh submit hw3
```

Academic Honesty

As for the previous assignment, for this assignment you may discuss with a partner, but you should write your own code; you should not be doing any pair programming or copying off one another's work. Similarly, please resist all temptation to take code that may be floating around in Penn or anywhere else. We will be checking all code against work from other students in the course, on top of other sources.

CIS 191: Linux and Unix Spring 2020

Homework 4

Due: Noon on Wednesday, April 8th, 2020.

A Linear Logistic Regression Classifier from Scratch in Python

Introduction

In this assignment you will use Python to create a command-line tool to train a binary linear logistic regression classifer and classify new data with the resulting classifier.

You will implement the classifier from scratch; DO NOT USE EXISTING, OUT-OF-THE-BOX CLASSIFIERS, such as the LogisticRegression class in the popular scikit-learn module. You may however use these classifiers to check your work.

I've provided a skeleton file containing functions and code blocks that you need to implement. This file will be your submission for this assignment. Do not let the length of the skeleton file intimidate you. There are only 7 functions and one code block to implement, and each of these can be implemented with a few lines of code, usually no more than 10. Most of the file is just specification comments!

If you don't already have Python 3 installed, install it, and the Python package manage pip like so,

```
apt install python3 pip3
```

then install the packages argparse, numpy, scipy, and matplotlib like so:

```
pip3 install argparse numpy scipy matplotlib
```

Though you are not required to use these packages, understanding how to use them will make your job much easier. More specifically the argparse module will make parsing command-line arguments relatively easy, numpy will make it easy for you to manipulate matrices, scipy.optimize will make it easy for you to optimize functions, and matplotlib.pyplot will make it easy for you to plot graphs. Also, the skeleton code is written in a way that encourages you to use these packages.

Machine Learning Background

A classifier is a program that takes a data point and labels it using a set of predetermined labels. A binary classifier uses two labels; for this assignment the two labels will be -1 and +1. The classifier that you implement will be trained on some data, after which the classifier can take an arbitrary data point that may or may not be in the training set and label it.

Mathematical Background

A vector of dimension d is simply a list of d numbers. Given d dimensional vectors u and v, the dot product $u \cdot v$ of u and v is the number:

$$u \cdot v = \sum_{i=1}^{n} u[i] \cdot v[i]$$

For example,

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

Linear Logistic Regression with Regularization

A linear logistic classifer classifies a d dimensional vector $\mathbf{v}$ using the formula

$$sign(\mathbf{c} \cdot \mathbf{v} + c_0)$$

for some d dimensional vector $\mathbf{c}$ and constant number c_0 that depend on training data.

More specifically, given training d dimensional training vectors $\mathbf{x}_1, \ldots \mathbf{x}_n$ with associated training labels, $l_1, \ldots, l_n$, $\mathbf{c}$ and c_0 are chosen to be the d dimensional vector $\mathbf{w}$ and constant number w_0 that minimize the following function over all d dimensional vectors:

$$\sum_{i=1}^{n} ln(1 + e^{-l_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)})$$

Slightly different objective functions that add *regularization* are often times used to reduce overfitting, which occurs when a function is too closely fit to a limited set of data points. Linear logistic regression with L_1 and L_2 regularization respectively lead to the following functions to be minimized instead,

$$\left\{ \sum_{i=1}^{n} ln(1 + e^{-l_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)}) \right\} + \lambda \cdot \sum_{i=1}^{d} |\mathbf{w}[i]|$$

$$\left\{ \sum_{i=1}^{n} ln(1 + e^{-l_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)}) \right\} + \lambda \cdot (\sqrt{\mathbf{w} \cdot \mathbf{w}})$$

for some regularization parameter λ, a number.

Command Line Arguments

The command

`./logistic_regression.py [-h] [-classify file] [-regularize n lambda] d learning_curve file_1 ... file_n`

should train a linear logistic classifer classifying d dimensional data using the data points in *file1* , ..., *file_n* to classify the data points in *file* , save a plot of the learning curve in *learning_curve* , and print the resulting labels line by line to standard output.

The dimension argument d is mandatory and must be an integer greater than 0.

The *learning_curve* option is mandatory and is the file to contain a plot of the *learning curve* . The learning curve is a curve showing the classification errors (on the y-axis) as a function of the number of training vectors used (on the x-axis) from 10% of the training data, 20% of the training data, and so on up to 100% of the training data. The *classification error* is the proportion of training data points classified incorrectly using the classifier relative to the number of all training data points. (Note: the training error should be calculated only on the subset of vectors used for training, not on all the training vectors available in the given data set.)

Each line in each training file *file_i* represents a training data point and will have $d + 1$ numbers separated by whitespace. The first d numbers represent the data point itself, with the last number in the line represents the training label associated to that data point. The entire training set is given by all the lines from all the training files. At least one valid non-empty training file must be provided.

The -classify option is optional. If it is provided then exactly one valid non-empty *file* must be provided for classification. Each line in *file* represents a data point to be classified and will have d numbers separated by whitespace.

The option -regularize is optional. If it is provided, then the regularization parameters n (1 or 2 for L_1 and L_2 regularization respectively) and λ must also be provided.

Your script should output a help message when run with the -h option. The help message should give a brief description of the script, as well as give a description of the intended usage and options.

Getting Some Data

For a start, you may use the training file train.txt which has 57 dimensional data and the classification file classify.txt to test your tool (this data is from here). If you wish to do some more testing, there are many dataset repositories out there. Some of the more popular ones are kaggle , the UC Irvine Machine Learning Repository and Amazon's Public Data Sets . Be aware that you probably will need to prepare these data sets to be compatible with your classifier. For instance, a data set may use labels in $\{0, 1\}$ instead of $\{-1, +1\}$.

Submission

For this assignment, please submit your `logistic_regression.py` script on eniac using `turnin .`

Academic Honesty

As for the previous assignment, for this assignment you may discuss with a partner, but you should write your own code; you should not be doing any pair programming or copying off one another's work. Similarly, please resist all temptation to take code that may be floating around in Penn or anywhere else. We will be checking all code against work from other students in the course, on top of other sources.

CIS 191 Spring 2020

Final Project: Introduction and Guidelines

Introduction

For the final project for CIS191, you will need to get your hands dirty with some kind of linux/unix project. This means you have a lot of flexibility, as long as your project has some kind of OS component.

> **NOTE**: You **MUST** work in groups of 2.

Below are some proposals, but feel free to suggest another idea. It may be the case that some of the ideas suggested on this page are not substantial enough on their own for the final project. You should judge the substance of the project idea using common sense (how much effort does a class project require?) and by conferring with Solomon.

Project Ideas

Option 1: Build a tool you will find useful

- Manage your data: photos, music, videos, etc
- Framework for managing your coursework or job search
- Calendar/to do list notification tool
- Package manager for a programming language or other domain
- Build automation tool (like Makefile)
- Dropbox-like backup tool

- An emacs mode or vim plugin

For this sort of project, you are encouraged to any tools and languages at your disposal, but should apply in some way to the skills you have learned in CIS191.

Option 2: Contribute to an open source project

Linux has a very close relationship with FOSS (Free and Open Source Software), and one option for the final project is to contribute to an open source project. You can browse for projects here or here. Some ways you can contribute might include:

- Add a feature
- Fix a bug that has been reported on the bug-tracking forums
- Package the project for distribution on various OSs (e.g. .deb for Debian-based systems, port to OSX, etc)
- Add documentation, examples, a tutorial, etc.
- Add support for Wine, to run windows executables on linux machines

Again, you are free to work with whatever tools or programming languages you like, and your project need not be based on an operating system, but your contribution should touch the OS in some way. For examle, fixing a bug might require you to replicate someone's OS configuration to find out how it is influencing the application.

Option 3: Investigate security issues

Break something or fix something that's broken! A good place to learn about hacks is from the ezine Phrack which written by and for hackers, and covers a wide range of topics including computer and physical security, hacking and cryptography. From here, you could, for example, learn how to Smash The Stack For Fun And Profit. For this project, ideally you would be able to perform an exploit to completion to demonstrate that you are indeed a hacker.

Option 4: Code your own kernel

In 1991, Linus Torvalds wanted to understand how the new intel i386 processor

worked, so he created a small kernel based on Tanenbaum's MINIX and uploaded it to the FTP server of the Finnish University and Research Network. To his surprise people showed a lot of interest, and before he knew it, Linux was born. You know how the story goes from here. Do you want to be the next Linus? Then code your own kernel!

There are lot's of OSDev websites out there to guide, perhaps the most accessible being James Molloy's kernel development tutorials. You (likely) won't get a legit working OS by the end of the semester, but you will learn a lot. For this project ideally your OS should set up virtual memory as well as a simple dynamic memory manager.

Project Requirements

You will be graded on the following criteria:

- Code quality (6 pts) - Does your code run? Does it crash? Does it have a nice interface?
- Project Size and Difficulty (6 pts) - Is your project significantly large? Remember that the project counts for twice the score of a single homework and you will be working in pairs, so the project as a whole should be 4 times as large as a single homework assignment.
- Best Practices (6 pts)
  - Your project **MUST** use version control
    - Commit often
    - Commit messages should be helpful
  - Your code should be modular
  - Comment your code!
  - Use whitespace appropriately
  - If it makes sense, you should use tools such as Make to make your work easier.
- Writeup (6 pts)
  - Members of your group
  - Description of your project

        - Specific components and milestones to complete
        - Contributions of both students
        - How to use your code (should also be included in a README)
        - Link to github
    - Give some background on the tools or design of your project
        - e.g. what similar tools exist and what are their plusses and minuses? (It's okay if yours implements strictly fewer features.)
- Presentation (6 pts)
    - Short (10 min) video presentation on the last day of class
    - Demo your project, able to answer questions

Submission and Presentation

> The writeup and code artifact for each group is due on the last day of classes (April 29 at noon).

On the last day of class we will have 10 minute video presentations from each group about your project. You should be prepared to show off your final product and also some of the internals of what you've done (code, etc).