STATS 607A (Fall '17): Assignment 3
Due: Oct 22, 2017 11:59 PM

Ambuj Tewari

Oct 13, 2017

This version: 1.0

How to turn in?

Zip the following 4 files:
`assignment_three_rmt.py`
`assignment_three_animate.py`
`assignment_three_fuel.py`
`assignment_three_answers.pdf`
into a single `.zip` file and name it `assignment_three_uniqname.zip` where `uniqname` is your U of M uniqname. Upload your zip file to canvas using the "Assignments" link on the left. Make sure you are submitting "Assignment 3".

The first 3 files should be python scripts that run without any error message. The last file should be a PDF file with answers to all questions below.

1 Verifying the Semicircle and Tracy-Widom Laws from Random Matrix Theory [RMT] (10 points)

In this problem, we will numerically verify two important RMT laws following the algorithmic recipes mentioned in Section 1 of the following article:
http://www-math.mit.edu/~edelman/publications/random_matrix.pdf
We will basically write the equivalent Python code for Algorithm 1 in the article linked to above.

The only difference is that we will NOT implement Algorithm 2 that computes the Tracy-Widom distribution. Instead, we will simply read a table containing pre-computed numerical values from the file: tracy-widom.csv

Download an (incomplete!) python script by following the following link:
assignment_three_rmt.py
Click on the "Raw" button on the top right and save the file as `assignment_three_rmt.py`.

Open the python script in your favorite text editor. You will see a bunch of places where a comment says `TASK x.y(.z)`. These are the places where you have to supply your own code.
Important: Please only change/add code where the tasks require you to do so (sometimes you'll have to replace a `pass` statement with real code, sometimes you'll be changing existing statements). Please *don't* modify the existing code and comments anywhere else! We might use automated scripts to grade your code and not following this suggestion will break those scripts.

We will now briefly describe your tasks. If something is unclear, please don't hesitate to email the instructor and/or GSI.

1.1 Sampling from GOE and computing eigenvalues (2 points)

Create a 2D $n \times n$ ndarray A (`a` in the code) whose entries are iid standard normal. Then set $S = (A + A^\top)/2$ (`s` in the code). The random matrix S is said to be drawn from the GOE (Gaussian Orthogonal Ensemble). GOE, GUE (the one we'll see in the next TASK) and GSE (Gaussian Symplectic Ensemble) are three classical random matrix ensembles. See, for example,
http://en.wikipedia.org/wiki/Random_matrix#Gaussian_ensembles
Once the symmetric matrix S has been generated, store its n real eigenvalues in a row of the 2D ndarray `v`.

TASK 1.1 has 2 subtasks: 1.1.1 and 1.1.2

1.2 Sampling from GUE and computing maximum eigenvalue (2 points)

Create a complex valued 2D $n \times n$ ndarray A (`a` in the code) whose entries are iid complex random variables $X + \iota Y$ where X, Y are independent standard normals and ι is the imaginary unit. Then set $S = (A + A^\star)/2$ (`s` in the code). Note that $A^\star$ denotes the Hermitian (or conjugate) transpose of the matrix A. The random matrix S is said to be drawn from the GUE (Gaussian Unitary Ensemble). Once the Hermitian matrix S has been generated, store its maximum (in value, not absolute value) among its n eigenvalues in an entry of the 1D ndarray `v1`.

TASK 1.2 has 2 subtasks: 1.2.1 and 1.2.2.

1.3 Normalize GOE eigenvalues (1 point)

The eigenvalues will be on an $O(\sqrt{n})$ scale. Divide `v` by $\sqrt{n/2}$ to get rid of the n dependence in the scale.

1.4 Plot the histogram of GOE eigenvalues against the theoretical prediction (1 point)

We will bin the eigenvalues in `v` using the function `numpy.histogram()`. We will use the following as bin boundaries:
`-2 -1.8 -1.6 ... 1.8 2`
The call to compute the histogram will result in two variable `hist` and `bin_edges`. You will have `len(bin_edges)` exactly 1 larger than `len(hist)` since the bin counts in `hist` are for eigenvalues that fell in the middle of two successive bin boundary values. Then plot the bin counts using `matplotlib.pyplot.bar()`. The code for plotting the semicircle theoretical prediction and for saving the file to a pdf document is already supplied.

TASK 1.4 has 2 subtasks: 1.4.1 and 1.4.2.

1.5 Normalize GUE max eigenvalues (1 point)

The largest eigenvalue will be distributed in a $O(n^{-1/6})$ sized band around its expected value of $O(\sqrt{n})$. So first subtract $2\sqrt{n}$ from `v1` and then multiply each entry by $n^{1/6}$.

1.6 Plot the histogram of GUE max eigenvalues against the theoretical prediction (1 point)

This TASK is similar to TASK 1.4 except that we bin `v1` instead of `v` and our bin boundaries will be:
`-5 -4.8 -4.6 ... 1.8 2`
You have to create the bar graph from numerical values in the (normalized) 1D array `v1`. The code to read in the theoretical Tracy-Widom prediction from the file `tracy-widom.csv` and to plot it has already been supplied.

TASK 1.6 has 2 subtasks: 1.6.1 and 1.6.2.

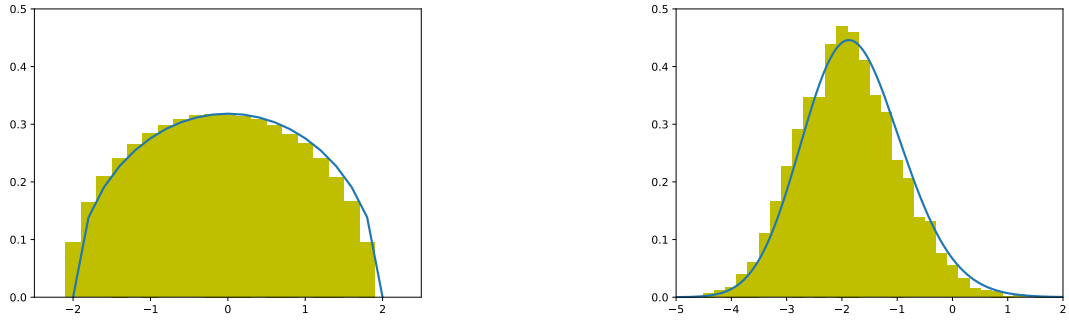

Figure 1: Numerical results versus theoretical predictions made by the Semicircle law for eigenvalue distribution of GOE ensembles (left) and Tracy-Widom law for maximum eigenvalue distribution of GUE ensembles (right)

1.7 Running the script and discussing the output (2 points)

Once you have supplied all missing pieces of the code, run the script using:
`python assignment_three_rmt.py`

Two pdf files `Semicircle.pdf` and `Tracy-Widom.pdf` will be created by the above script. The figures in them should like the ones shown in Figure 1.

Answer the following questions:

Q. 1.1 There are several eigenvalue related functions available in `scipy.linalg` including `eig, eigh, eigvals, eigvalsh`. How did you decide which one to choose? Did you look at speed? Applicability to this problem? Anything else?

Q. 1.2 Do you still get the same plots as in Figure 1 if you replace all standard normals you used in the sampling stage with symmetric Bernoulli random variables (i.e., discrete random variables that are either $+1$ to -1 with equal probability).

2 Creating animation videos for online learning algorithms (7 points)

In this problem, we will create short video animations for two online learning algorithms for binary classification: perceptron and online logistic regression. Both algorithms start with a weight vector $w \in \mathbb{R}^d$ (d will be equal to 2 so that we can plot everything in 2D). They both process the data one example, label pairs at a time.

Given an example $x \in \mathbb{R}^d$ and a label $y \in \{\pm 1\}$, perceptron first tests whether $yw^\top x > 0$. If it is, then no update is made. If not, then it sets w to $w + yx$.

Given an example $x \in \mathbb{R}^d$ and a label $y \in \{\pm 1\}$, online logistic regression performs the update

$$w = w - \eta \ell'(w^\top x, y)x$$

where $\ell'(t, y)$ is the derivative of $\ell(t, y) = \log(1 + \exp(-yt))$ w.r.t. t. The step size parameter η has been set to its default value of 0.1 in our implementation. The python code for computing the derivative of the logistic loss is already provided in the file:
https://github.com/ambujtewari/stats607a-fall2017/blob/master/homeworks/losses.py
which has already been imported for you in the file for this problem.

Download an (incomplete!) python script by following the following link:
assignment_three_animate.py
Click on the "Raw" button on the top right and save the file as `assignment_three_animate.py`.

2.1 Perceptron update (1 point)

Implement the function `perceptron_update`.

2.2 Online logistic regression update (1 point)

Implement the function `online_lr_update`.

2.3 Create the basic plots (1.5 points)

Provide code to plot the positives as blue dots, negative as red dots, the separator line (whose equation is $w^\top x = 0$) as a black line.

This TASK consists of 3 subtasks: 2.3.1 through 2.3.3

2.4 Fill in the frame update (1.5 points)

The animation basically works by repeated calling `frame_update` to change the plot (each time with a different argument). We have set up the animation so that `frame_update` will get called with argument drawn from the list `range(2*n+1)`. We won't do anything on the 0th call. On all subsequent calls, we will execute one update of whichever algorithm we're animating. The example picked at iteration `i` will be `(i-1) % n` (along with the corresponding label). Then we will update the separator (black) line and also the current point marked with a yellow star.

This TASK consists of 3 subtasks: 2.4.1 through 2.4.3

2.5 Running the script and discussing the output (2 points)

Once you have supplied all missing pieces of the code, run the script using:
`python assignment_three_animate.py`

This will create two `mp4` files in the current directory:
`Online_logistic_regression_anim.gif` `Perceptron_anim.gif`
These should be similar to these ones posted on the course website:
Online_logistic_regression_anim.gif
Perceptron_anim.gif
Open the files created by your python script in your favorite browser and answer the following questions:

Q. 1.1 Perceptron runs on data that is perfectly separable using a (not necessarily unique) linear separator. Does it find one such separator? If not, how many points does the final classifier misclassify? If we keep running the algorithm by cycling through the data, will it eventually classify everything correctly?

Q. 1.2 Online logistic regression runs on non linearly-separable data. How many points does the final classifier misclassify? If we keep running the algorithm by cycling through the data, will it eventually classify everything correctly?

3 Fetching Alternative Fuel Stations Data via the NREL Web API (8 points)

In this problem, we will fetch data from the NREL (National Renewable Energy Laboratory) web API described at:

https://developer.nrel.gov/docs/transportation/alt-fuel-stations-v1/nearest/
Familiarize yourself with how the API works by browsing the above website. Note that you will have to fill out a form to receive your own private API key:
https://developer.nrel.gov/docs/api-key/
Without this 40 character string you will not be able to finish this problem.

We will store the alternative fuels data for Ann Arbor in a Pandas frame and do some simple computations with it.

Download an (incomplete!) python script by following the following link:
assignment_three_fuel.py
Click on the "Raw" button on the top right and save the file as `assignment_three_fuel.py`.

3.1 Obtain the API key and store it in a variable (1 point)

Use the web form above to request an API key. The key will be sent to you via email. Store the 40 character key in the variable `api_key`.

3.2 Construct the query URL (1 point)

The variables `location` and `query_fmt` have been defined for you. Use them along with `api_key` to construct the query URL and store it in the variable `url`.

3.3 Getting the total number of results (1 point)

The query is executed by the function `get_data` that has already been defined for you. It converts data from the JSON format to a Python dict which is then stored in a variable `data`. There is one key in this dict that will give you the number of total results. Store that number, as an `int`, in the variable `nresults`.

3.4 Getting the initial list of fuel station data (1 point)

Another key in the dict `data` has fuel station information. Store that list in the variable `fs_list`.

3.5 Getting data from all remaining fuel stations (2 points)

The number of fuel stations about which we have data in `fs_list` may not be equal to `nresults`. You will need to fetch additional results using the "offset" parameter as described in the web API documentation. Each time you fetch additional results, you should grow the list `fs_list` to include the additional results. Stop when the list `fs_list` has length equal to `nresults`.

This TASK consists of 2 subtasks: 3.5.1 and 3.5.2

3.6 Create a DataFrame with information about fuel stations (1 point)

Convert the list `fs_list` into a pandas DataFrame `df`. The supplied code prints the number of fuel station attributes and the number of fuel stations. It also print a few of the interesting columns of the DataFrame.

3.7 Running the script and discussing the output (1 point)

Once you have supplied all missing pieces of the code, run the script using:
`python assignment_three_fuel.py`

The script will produce some text output first about its progress and then about the size of the data frame created.

Answer the following questions:

Q. 3.1 Is the number of rows in `df` the same as `nresults`? If not, why do you think these two numbers are different?

Q. 3.2 How many alternative fuel types are available in Ann Arbor? Which alternative fuel type is most common among Ann Arbor alternative fuel stations?