Homework 3: Histograms and Distributions

Due: 02.12.2021 at 11:59 pm ET

This homework asks you to write and use the functions we discussed in the lecture on histograms to find the "optimal" number of bins for a data set. We then ask you to explore some data to identify its distribution.

Goals

In this homework you will:

1. Learn how to install new Python modules
2. Build up a complex analysis code by building smaller functions first
3. Perform some basic data exploration

Background

Python modules

Python has a lot of built-in functionality right out of the box: basic data structures like lists, sets, and dictionaries, functions that help use those data structures (like `len`), etc. There are a vast number of Python modules that provide additional functionality too. This functionality is not built in -- not everyone needs it -- but Python comes with tools to make it easy to use those modules.

Installing a module

To install a module, you can use Python's built-in package manager, `pip`. We will provide instructions for installing modules from the command line. These instructions will work with Python3 installations on Scholar, ECE Grid, most Linux distributions, and Mac OS. There are a number of different ways to get Python3 on Windows, so you will have to look at the documentation for your version to determine how to install a new module.

Modules can be installed globally (so everyone on a machine has access to them) or locally (so only you have access to them). Installing modules globally requires root access to the machine (or other specially-set permissions), so we will provide instructions for installing modules locally.

To install a module named `foo`, you can use the following command:

```
python3 -mpip install --user foo
```

This commend can let you install the latest version of the package. Add you can also install an exact version of the package by adding the version number after the package name followed by the sign `==`:

```
python3 -mpip install --user foo==2.0.8 # install the foo package with version number 2.0.8
```

If `foo` is already installed on your system but you want to upgrade to a new version of it, you can use the command:

```
python3 -mpip install --user --upgrade foo
```

There are also other choices of package manager, for example if you are using `conda` to manage your python environment, you can do the same thing as `pip` does with the following commands:

```
conda install foo # Install the latest version of foo package
```

```
conda install foo=2.0.8 # Install the foo version 2.0.8
```

```
conda update foo # Update the package foo
```

Note: In order to specify a specific version number `pip` uses `==` while `conda` uses `=` .

To complete this homework, you will need to install the following modules:

1. `numpy==1.14.0` : this is a module that provides array and matrix classes, and many mathematical operations on those classes. It is the foundation of many of the modules that are used in data science.
2. `scipy==1.0.0` : this module provides many other useful functions for data analysis, including functions for dealing with probability distributions
3. `matplotlib==2.1.2` : a basic plotting/visualization library

Note: You are welcome to use different versions of packages since a lot of them may already been installed in your environment. There might be a slight chance that potential problems will occur if you are using packages that are in a different version especially a much older version then provided above. Therefore, please make sure your version is at least the version number provided above.

Using a module

The functions in a module are in that module's *namespace*. To make sure that the function names do not collide with functions in other modules (or Python's built-in functions), the functions need to be accessed through a prefix. To load a module, you have to tell Python (a) which module to load; and (b) what prefix to use when accessing the functions of that module. For example, the following code:

```
import numpy as np
```

Tells Python you want to use the module `numpy` , and that you want to access the functions of `numpy` using the prefix `np` . For example, the following code will read in a list of numbers stored in a text file and give you back a list with those numbers in it:

```
data = np.loadtxt('input.txt')
```

You can also use the `import` keyword to bring in functions from other files (think of these like `#include` directives in C). The following command will import the `function1` you wrote in myfile (if it is in a file called `myfile.py`):

```
from myfile import function1
```

You can also import all of the functions from another file (`eg: helper.py`):

```
from helper import *
```

Incremental Development

In this homework, we will ask you to write a fairly complex piece of code: finding the number of histogram bins that results in the lowest error for a given data set. When you need to write complex code like this, your goal should be to break the problem down into smaller pieces. Write functions that solve each of the smaller pieces, then figure out how to connect those functions together (some of them might call other functions you write) to solve the overall problem.

This approach makes it much easier to write complex code, both because you do not have to solve the problem all in one go, and because it makes it easier to *test* your code: you can test each of your smaller pieces individually to make sure that they work properly.

In this homework, we will walk you through one particular way you can break down the problem (and, in fact, we want you to solve the problem in this way -- we will test the individual pieces for partial credit). Please be mindful that variables declared in a function can only be accessed when that particular function is being called.

Instructions

0) Set up your repository for this homework

Click the link on Piazza to set up your repository for HW 3, then clone it.

The repository should contain 8 files:

1. This README
2. 4 input data files: `input.txt` , which you will use in problem1, and `distA.csv` , `distB.csv` , and `distC.csv` , which you will use in problem 2
3. A helper file, `helper.py` that contains some useful helper functions for use when writing/testing your code
4. `problem1.py` , a skeleton script for Problem 1
5. `testbin.py` , to help individually test the functions you write in `problem1.py`

1) Problem 1: Histogram Bin Width Optimization

In this problem, you will implement histogram bin width optimization from a data set. You will use the histogram function in `matplotlib.pyplot` , accessed as `matplotlib.pyplot.hist` or `plt.hist` if you `import matplotlib.pyplot as plt` . Please read the documentation: `matplotlib.pyplot.hist` . (Note in particular that the function returns three variables: `n` , `bins` , and `patches` , but you only need `n` , so be sure to unpack the output accordingly.)

We have broken the problem down into smaller pieces for you. `problem1.py` has four functions for you to fill in. **Keep the signatures of these functions the same as you are filling them in; we will use these to assign partial credit**.

1. `norm_histogram` takes a histogram of counts and creates a histogram of probabilities.
2. `compute_j` computes the value of J for a given histogram and bin width.
3. `sweep_n` computes the `compute_j` score for each of a range of *numbers of buckets*and returns a list of the associated `compute_j` scores. This function should use `compute_j` in its implementation (this function should use `matplotlib.pyplot.hist` in its implementation. Note that `sweep_n` cares about the number of buckets while `compute_j` cares about the width of the buckets -- make sure to do the conversion!).
4. `find_min` is a generic function that takes a list of numbers and returns a tuple containing the smallest number in that list and the index of that smallest number.

You can use `input.txt` , provided in the repository, as test data. To test each function individually please refer to `testbin.py` . There are instructions available to test each portion of your code.

Within `testbin.py` if:

1. `norm_histogram` runs correctly then the output (shown only up to 3 points post decimal) will be `['0.008', '0.036', '0.097', '0.188', '0.254', '0.216', '0.130', '0.043', '0.021', '0.007']`
2. `compute_j` works then the output will be `-0.026`
3. `sweep_n` works then the output (shown only up to 3 points post decimal) should be `['-0.016', '-0.016', '-0.026', '-0.025', '-0.026', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.028', '-0.028', '-0.028', '-0.028', '-0.027', '-0.028', '-0.027', '-0.028', '-0.028', '-0.028', '-0.027', '-0.028', '-0.028', '-0.027', '-0.028', '-0.028', '-0.027', '-0.028', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.028', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.026', '-0.027', '-0.026', '-0.027', '-0.026', '-0.027', '-0.027', '-0.027', '-0.027', '-0.027', '-0.026', '-0.027', '-0.026', '-0.027', '-0.027', '-0.026', '-0.026', '-0.027']`
4. `find_min` executes then your output should be `(-0.027714308366937675, 16)`

Please note: you are not to truncate the values. We have only done so to keep the write up brief. The output from `find_min` will be checked up to three points post decimal.

If your functions all work, and you run the test code that is included in `problem1.py` , you should produce the following output: `(-0.027714308366937675, 16)`

The expected outputs show only up to three points of precision, your result from running `testbin.py` may contain longer floating points.

> The `if __name__ == '__main__'` line in `problem1.py` is a useful way to write tests for your code: this is code that will *only* run if you run this file as the main script; if this file is included from another script, this test code will not run.

Problem 2: Distributions

For problem 2, put your code in a file called `problem2.py` and your writeup in a file called `problem2.pdf`. Only in this problem you may use `jupyter notebook` and have your writeup and plots in one file titled `problem2.ipynb`.

Problem 2 of the homework asks you to compute quantile-quantile (QQ) plots for three input data sets: `distA.csv`, `distB.csv`, `distC.csv`.

Each of these datasets was generated using one of 8 possible distributions:

- Gaussian (`norm`)
- Cauchy (`cauchy`)
- Cosine (`cosine`)
- Exponential (`expon`)
- Uniform (`uniform`)
- Laplace (`laplace`)
- Wald (`wald`)
- Rayleigh (`rayleigh`)

For each dataset, tell us which distribution was used to generate the data (you may find it helpful to plot histograms of the datasets). Use QQ plots as part of your answer. You may find the `scipy` function `probplot` useful for this. For example, the following code will create a QQ plot comparing an input data set to a Gaussian distribution:

```
import scipy.stats as stats
import matplotlib.pyplot as plt
import numpy as np

data = getData(`distA.csv`)

stats.probplot(data, dist = 'norm', plot=plt)
plt.show() # modify this to write the plot to a file instead
```

(To compare to the other distributions, use the names in parentheses from the above list)

To write plots to a file, use `matplotlib.pyplot.savefig`

For each dataset, save the QQ plot to a file called `[dataset]-[dist].png` (or keep the figure inline if you are using Jupyter Notebook), where [dataset] is the name of the data set (e.g., `distA`) and [dist] is the name of the distribution. In your writeup, tell us, for each data set, *which distribution you think matches that data set*. You may not use more than two full sentences to describe which distribution best fits your QQ plot.

What to Submit

For Problem 1, please submit `problem1.py` with all the appropriate functions filled in.

For Problem 2, please submit either `problem2.pdf` or `problem2.ipynb` which should contain (i) your best fit plot for each dataset, and (ii) a one or two sentence explanation describing why each plot is the best fit. Please do not use any other format for submission.

Submitting your code

Please commit the latest version of your code as you did in HW2. Do not make further modifications to your repository.