QMB 3311: Python for Business Analytics
Department of Economics
College of Business
University of Central Florida
Spring 2022

Assignment 7

Due Sunday, April 17, 2022 at 11:59 PM
in your GitHub repository

Instructions:

Complete this assignment within the space on your private GitHub repo in a folder called `assignment_07`. In this folder, save your answers to Questions 1 and 2 in a file called `my_logit_A7.py` as described in Question 1. In the same folder, submit the script `logit_calculation_A7.py` for Question 3. When you are finished, submit it by uploading your files to your GitHub repo using any one of the approaches outlined in Question 4. You are free to discuss your approach to each question with your classmates but you must upload your own work.

Question 1:

Module `my_logit_A7.py`:
Collect the following functions from your previous assignments into a module called `my_logit_A7.py`.

- `logit()` from Assignment 3.
- `logit_like()` from Assignment 3.
- `logit_like_sum()` from Assignment 4.
- `logit_di()` from Assignment 5.
- `logit_dLi_dbk()` from Assignment 5.

Testing:
For all of the Exercises in Question 2, and the functions listed above, use examples to test the functions you defined. Since the examples will all be contained within the docstrings of your functions, you can use the `doctest.testmod()` function within the `doctest` module to test your functions automatically at the bottom of the modules. Use an `if` statement to determine whether the tests will be run using `testmod`: if the script is executed as a script, i.e. as the `__main__` script, then run the tests; if it is imported, then skip the tests.

Question 2:

Follow the function design recipe to define functions for all of the following Exercises. For each function, create three examples to test your functions. Record the definitions in the sample script `my_logit_A7.py`.

Example 1 Write a function `logit_like_sum_opt(beta, y, x)` that is a wrapper for the function called `logit_like_sum()` that can be used for optimization. The parameter for optimization `beta` should be a single parameter containing both `beta_0` and `beta_1` and it should be the first parameter. Also, the function should return the negative of `logit_like_sum()` since the `scipy` algorithms minimize the objective function and the estimates of `beta` are those that maximize `logit_like_sum()`.

Example 2 Write a function `logit_like_grad(beta, y, x)` to calculate the gradient vector of the likelihood function `logit_like_sum_opt()`. It should take the same arguments as the function `logit_like_sum_opt()`, in the same order. The *gradient vector* of a multivariate function is a vector with each element equal to the derivative of the function with respect to each parameter. In the case of $L(y, x; \beta_0, \beta_1)$, element k is

$$\frac{\partial L(y, x; \beta_0, \beta_1)}{\partial \beta_k} = \sum_{i=1}^{n} \frac{\partial}{\partial \beta_k} L_i(y_i, x_i; \beta_0, \beta_1)$$

for $k = 0$ or $k = 1$, corresponding to β_0 or β_1, where $L_i(y_i, x_i; \beta_0, \beta_1)$ is defined in Exercise 2 above.

Using calculus, one can determine that

$$\frac{\partial}{\partial \beta_k} L_i(y_i, x_i; \beta_0, \beta_1) = \begin{cases} d_i(1 - \ell(x_i; \beta_0, \beta_1)), & \text{if } y_i = 1, \\ d_i(-\ell(x_i; \beta_0, \beta_1)), & \text{if } y_i = 0, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where

$$d_i = \begin{cases} 1, & \text{for } k = 0, \\ x_i, & \text{for } k = 1. \end{cases}$$

We coded these functions in Assignment 5. Using these expressions, you can define this function `logit_like_grad()` with the sum of the calculations of `logit_dLi_dbk(y_i, x_i, k, beta_0, beta_1)` from Assignment 5. Your function will output a vector of two elements, corresponding to the parameters β_0 (for `k = 0`) and β_1 (for `k = 1`). Like the objective function above, this new function will take the *negative* of the sum of the terms in `logit_dLi_dbk(y_i, x_i, k, beta_0, beta_1)`, where `i` loops over the elements in the lists `y` and `x`.

Question 3:

The sample script `logit_calculation_A7.py` uses the `statsmodels` module to estimate a model for the probability that borrowers will default on their loans. You will calculate the parameter estimates by applying optimization methods in `scipy` to estimate these parameters. The dataset `credit_data.csv` in `demo_19_Classification` includes the following variables.

default:	1 if borrower defaulted on a loan
bmaxrate:	the maximum rate of interest on any part of the loan
amount:	the amount funded on the loan
close:	1 if borrower takes the option of closing the listing until it is fully funded
AA:	1 if borrowers FICO score greater than 760
A:	1 if borrowers FICO score between 720 and 759
B:	1 if borrowers FICO score between 680 and 719
C:	1 if borrowers FICO score between 640 and 679
D:	1 if borrowers FICO score between 600 and 639

In the script `logit_calculation_A7.py`, these data are loaded in and used to estimate a model using `statsmodels`, with only the variable `bmaxrate`. We use the functions defined above in exercise 1 and 2 above. The function `logit_like_sum_opt(beta, y, x)` is the (negative of the) log-likelihood function that is maximized to get the parameter estimates in `statsmodels`. The function `logit_like_grad(beta, y, x)` is the (negative of the) first derivative of the log-likelihood function, which is zero at the maximal parameter values. **No examples are necessary, since you have already tested the functions. All you need to do is obtain the coefficients by optimization, filling in the code in `logit_calculation_A7.py` wherever it is marked `Code goes here`.**

a) Run the script `logit_calculation_A7.py` up to line 150 to see the results for the estimation with `statsmodels`. The goal is to match the parameter estimates in `logit_model_fit_sm.params` and achieve the maximum value of the log-likelihood function shown in the output from `logit_model_fit_sm.summary()`.

b) Calculate the parameter estimates by minimizing `logit_like_sum_opt(beta, y, X)` using the function `minimize()` from the `scipy` module and passing the tuple of arguments `(y, X)`. Implement it several times using the following algorithms.

   i) Use the Nelder-Mead Simplex algorithm algorithm by passing the argument `method = 'nelder-mead'`.

   ii) Use the Davidon-Fletcher-Powell (DFP) algorithm by passing the argument `method = 'powell'`.

   iii) Use the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS) algorithm by passing the argument `method = 'BFGS'`.

   iv) Use another version of the BFGS algorithm. This time, pass the additional argument `jac = logit_gradient` to use the first derivative to calculate the iterations within the algorithm.

c) Verify that your parameter estimates and the optimal values of the likelihood function are achieved with the methods in part (b), to match the results from `statsmodels`. You may need to pass additional arguments to the `options` argument, as in:

```
options = {'xtol': 1e-8, 'maxiter': 1000, 'disp': True}
```

and to adjust the values as necessary. Compare the accuracy and number of iterations.

Question 4:

Push your completed files to your GitHub repository following one of these three methods.

Method 1: In a Browser

Upload your code to your GitHub repo using the interface in a browser.

1. Browse to your `assignment_0X` folder in your repository (the "X" corresponds to Assignment X.).
2. Click on the "Add file" button and select "Upload files" from the drop-down menu.
3. Revise the generic message "Added files via upload" to leave a more specific message. You can also add a description of what you are uploading in the field marked "Add an optional extended description..."
4. Press the button "'Commit changes," leaving the buton set to "Commit directly to the `main` branch."

Method 2: With GitHub Desktop

Upload your code to your GitHub repo using the interface in GitHub Desktop.

1. Save your file within the folder in your repository within the folder referenced in GitHub Desktop.
2. When you see the changes in GitHub Desktop, add a description of the changes you are making in the bottom left panel.
3. Press the button "Commit to main" to commit those changes.
4. Press the button "Push origin" to push the changes to the online repository. After this step, the changes should be visible on a browser, after refreshing the page.

Method 3: At the Command Line

Push your code directly to the repository from the command line in a terminal window, such as GitBash on a Windows machine or Terminal on a Mac.

1. Open GitBash or Terminal and navigate to the folder inside your local copy of your git repo containing your assignments. Any easy way to do this is to right-click and open GitBash within the folder in Explorer. A better way is to navigate with UNIX commands, such as `cd`.

2. Enter `git add .` to stage all of your files to commit to your repo. You can enter `git add my_filename.ext` to add files one at a time, such as `my_functions.py` in this Assignment.

3. Enter `git commit -m "Describe your changes here"`, with an appropriate description, to commit the changes. This packages all the `add`ed changes into a single unit and stages them to `push` to your online repo.

4. Enter `git push origin main` to push the changes to the online repository. After this step, the changes should be visible on a browser, after refreshing the page.