HW8: Linear Regression

Due Nov 17 by 11:05am **Points** 100 **Submitting** a file upload **File Types** py
Available Nov 10 at 11am - Nov 17 at 11:05am 7 days

This assignment was locked Nov 17 at 11:05am.

Change Log

- 11/12/2020 Clarify return value of gradient_descent: should be an 1D array of partial derivatives
- 11/12/2020 One more example output for SGD
- 11/11/2020 Add sample output for compute_betas
- 11/10/2020 Clarify get_dataset output

Assignment Goals

- Implement a linear regression calculation
- Examine the trends in real (messy) data

Summary

Percentage of body fat, age, weight, height, and ten body circumference measurements (e.g., abdomen) are recorded for 252 men. Body fat, one measure of health, has been accurately estimated by an underwater weighing technique. Fitting body fat to the other measurements using multiple regression provides a convenient way of estimating body fat for men using only a scale and a measuring tape. In this assignment, you will be looking at the **bodyfat dataset (http://jse.amstat.org/v4n1/datasets.johnson.html)** and build several models on top of it.

Program Specification

You will be using the bodyfat dataset (**bodyfat.csv**) for this assignment. Complete the following Python functions in this template `regression.py`:

1. **get_dataset(filename)** — takes a filename and **returns** the data as described below in an n-by-(m+1) array
2. **print_stats(dataset, col)** — takes the dataset as produced by the previous function and **prints** several statistics about a column of the dataset; does not return anything
3. **regression(dataset, cols, betas)** — calculates and **returns** the mean squared error on the dataset given fixed betas
4. **gradient_descent(dataset, cols, betas)** — performs a single step of gradient descent on the MSE and **returns** the derivative values as an 1D array

5. **iterate_gradient(dataset, cols, betas, T, eta)** — performs T iterations of gradient descent starting at the given betas and **prints** the results; does not return anything
6. **compute_betas(dataset, cols)** — using the closed-form solution, calculates and **returns** the values of betas and the corresponding MSE as a tuple
7. **predict(dataset, cols, features)** — using the closed-form solution betas, **return** the predicted body fat percentage of the give features.
8. **sgd(dataset, cols, betas, T, eta)** — performs stochastic gradient descent, **prints** results as in function 5

Get Dataset

The `get_dataset()` function should return an n-by-(m+1) array of data, where n is the number of data points, and m is the number of features, plus an additional column of labels. The first column should be bodyfat percentage, which is the **target** that our regression model aims for. We denote it as **y** in the rest of the write up. Starting from the second column, there goes a list of **features**, including density, age, weight, and more. We use index 1 to represent density, 2 to represent age, and so on... You should ignore the "IDNO" column as it merely represents the individual id of each participant.

```
>>> get_dataset('bodyfat.csv')
=> array([[12.6 , 1.0708, 23. , ..., 32. , 27.4 , 17.1 ],
          [ 6.9 , 1.0853, 22. , ..., 30.5 , 28.9 , 18.2 ],
          [24.6 , 1.0414, 22. , ..., 28.8 , 25.2 , 16.6 ],
          ...,
          [28.3 , 1.0328, 72. , ..., 31.3 , 27.2 , 18. ],
          [25.3 , 1.0399, 72. , ..., 30.5 , 29.4 , 19.8 ],
          [30.7 , 1.0271, 74. , ..., 33.7 , 30. , 20.9 ]])
>>> dataset = get_dataset('bodyfat.csv')
>>> dataset.shape
(252, 16)
```

Dataset Statistics

This is just a quick summary function on one **feature**, given in the parameter col, from the above dataset. When called, you should **print**:

1. the number of data points
2. the sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
3. the sample standard deviation $\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$

on three lines. Please format your output to include only **TWO digits** after the decimal point. For example:

```
>>> data = get_dataset('bodyfat.csv')
>>> print_stats(dataset, 1)  # summary of density
252
```

```
1.06
0.02
```

You might find **this guide** (https://pyformat.info/) to python print formatting useful.

Linear Regression

This function will perform linear regression with the model

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m$$

We first define the mean squared error (MSE) as the sum of squared errors divided by # data points:

$$MSE(\beta_0, \beta_1, \cdots, \beta_m) = \frac{1}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} - y_i)^2$$

The first argument refers to the dataset, and the second argument is a list of features that we wish to learn on. For example, if we would like to study the relationship of body fat vs age and weight, cols should be [2, 3]. In this case, the model should be $f(x) = \beta_0 + \beta_2 x_2 + \beta_3 x_3$. The last argument (betas) of this function represents three betas, $\beta_0,\ \beta_2,\ \beta_3$. **Return** the corresponding MSE as calculated on your dataset.

```
>>> regression(dataset, cols=[2,3], betas=[0,0,0])
=> 418.50384920634923
>>> regression(dataset, cols=[2,3,4], betas=[0,-1.1,-.2,3])
=> 11859.17408611111
```

Gradient Descent

This function will perform gradient descent on the MSE. At the current parameter $(\beta_0, \beta_1, \ldots, \beta_m)$, the gradient is defined by the vector of partial derivatives:

$$\begin{aligned}
\frac{\partial \mathrm{MSE}(\beta_0, \beta_1, \ldots, \beta_m)}{\partial \beta_0} &= \frac{2}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} - y_i) \\
\frac{\partial \mathrm{MSE}(\beta_0, \beta_1, \ldots, \beta_m)}{\partial \beta_1} &= \frac{2}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} - y_i) x_{i1} \\
\ldots
\end{aligned}$$

This function **returns** the corresponding gradient as a 1-D numpy array with the partial derivative with respect to β_0 as the first value.

```
>>> gradient_descent(dataset, cols=[2,3], betas=[0,0,0])
=> array([ -37.87698413, -1756.37222222, -7055.35138889])  # order: [partial derivative of beta_0, beta_2, be
ta_3]
```

Iterate Gradient

Gradient descent starts from initial parameter $\left(\beta_0^{(0)}, \beta_1^{(0)}, \ldots, \beta_m^{(0)}\right)$ and iterates the following updates at time t = 1, 2, ... , T:

$$\beta_0^{(t)} = \beta_0^{(t-1)} - \eta \frac{\partial \mathrm{MSE}\left(\beta_0^{(t-1)}, \beta_1^{(t-1)}, \ldots, \beta_m^{(t-1)}\right)}{\partial \beta_0}$$

$$\beta_1^{(t)} = \beta_1^{(t-1)} - \eta \frac{\partial \mathrm{MSE}\left(\beta_0^{(t-1)}, \beta_1^{(t-1)}, \ldots, \beta_m^{(t-1)}\right)}{\partial \beta_1}$$

and so on for the rest.

The parameters to this function are the dataset and a selection of features, the T number of iterations to perform, and η (eta), the parameter for the above calculations. Begin from the initial value as specified in the parameter betas.

Print the following for each iteration on one line, separated by spaces:

1. the current iteration number beginning at 1 and ending at T
2. the current MSE
3. the current value of beta_0
4. the current value of other betas

As before, all floating point values should be **rounded** to **two digits** for output.

```
>>> iterate_gradient(dataset, cols=[1,8], betas=[400,-400,300], T=10, eta=1e-4)
1 423085332.40 394.45 -405.84 -220.18  # order: T, mse, beta0, beta1, beta8
2 229744495.73 398.54 -401.54 163.14
3 124756241.68 395.53 -404.71 -119.33
4 67745350.04 397.75 -402.37 88.82
5 36787203.39 396.11 -404.09 -64.57
6 19976260.50 397.32 -402.82 48.47
7 10847555.07 396.43 -403.76 -34.83
8 5890470.68 397.09 -403.07 26.55
9 3198666.69 396.60 -403.58 -18.68
10 1736958.93 396.96 -403.20 14.65
```

Try different values for eta and a much larger T, and see how small you can make MSE (optional).

Compute Betas

Instead of using gradient descent, we can compute the closed-form solution for the parameters directly. For ordinary least-squares, this is

$$\hat{\beta} = (X^T X)^{-1} X^T \mathbf{y}$$

This function returns the calculated betas (selected by cols) and their corresponding MSE in a **tuple**, as (MSE, beta_0, beta_1, and so on).

```
>>> compute_betas(dataset, cols=[1,2])
=> (1.4029395600144443, 441.3525943592249, -400.5954953685588, 0.009892204826346139)
```

Predict Body Fat

Using your closed-form betas, predict the body fat percentage for a given number of features. Return that value.

For example:

```
>>> predict(dataset, cols=[1,2], features=[1.0708, 23])
=> 12.62245862957813
```

Stochastic Gradient Descent

Now let's have some fun with randomness and implement Stochastic Gradient Descent (SGD). With everything the same as part 5, modify the gradient as follows: in iteration t, randomly pick ONE of the n items (call it $\left(x_{j1_t}, x_{j2_t}, \ldots, x_{jm_t}, y_{j_t}\right)$), and approximate the gradient using *only that item*.

$$\frac{\partial \mathrm{MSE}(\beta_0,\beta_1,\ldots,\beta_m)}{\partial \beta_0} \approx 2(\beta_0 + \beta_1 x_{j1_t} + \cdots + \beta_m x_{jm_t} - y_{j_t})$$

$$\frac{\partial \mathrm{MSE}(\beta_0,\beta_1\ldots,\beta_m)}{\partial \beta_1} \approx 2(\beta_0 + \beta_1 x_{j1_t} + \cdots + \beta_m x_{jm_t} - y_{j_t}) x_{j1_t}$$

Note that you **must** use the provided function `random_index_generator` with **seed 42** to complete the selection step, for grading purposes. **Print** the same information as in the previous function. For example:

```
>>> sgd(dataset, cols=[2,3], betas=[0,0,0], T=5, eta=1e-6)
1 387.33 0.00 0.00 0.00
2 379.60 0.00 0.00 0.01
3 335.99 0.00 0.00 0.01
4 285.89 0.00 0.00 0.02
5 245.75 0.00 0.01 0.03
>>> sgd(dataset, cols=[2,8], betas=[-40,0,0.5], T=10, eta=1e-5)
1 123.77 -40.00 0.01 0.52
2 105.37 -40.00 0.01 0.53
3 72.07 -40.00 0.02 0.55
4 50.75 -40.00 0.02 0.57
5 29.90 -40.00 0.04 0.58
6 21.53 -40.00 0.05 0.60
7 20.57 -40.00 0.05 0.60
8 20.90 -40.00 0.05 0.62
9 19.96 -40.00 0.05 0.62
10 20.09 -40.00 0.05 0.62
```

Since n is small in our dataset, there is little advantage of SGD over standard gradient descent. However, on large datasets, SGD becomes much more desirable...

Discovery

Experiment on different combinations of features, and learning methods (closed form solution, gradient descent, or SGD). Try to answer these questions:

- What is the best feature that predicts well?
- What are the best set of features that predict well?
- Which learning method is the most effective in terms of training error (MSE)?
- Which learning method is the most efficient (takes the least time)?
- What will happen if our dataset has more than 1 million entries?

We won't grade these questions, but feel free to share your thoughts on Piazza!

Submission

Please submit your code in a file called `regression.py`. All code should be contained in the provided functions, or your helper functions, or under a

```
if __name__=="__main__":
```

check so that it will not run if your code is imported to another program.

Be sure to **remove all debugging output** before submission. Failure to remove debugging output will be **penalized (10pts)**.

If a regrading request isn't justifiable (the initial grade is correct and clear, subject to the instructors' judgment), the request for regrading will be **penalized (10 pts)**.

This assignment due at 11/17/2020 11:00am. Submitting right at 11:00am will result in a late submission. It is preferable to first submit a version well before the deadline (at least one hour before) and check the content/format of the submission to make sure it's the right version. Then, later update the submission until the deadline if needed.

HW8

Criteria	Ratings		Pts
get_dataset	**10.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
print_stats	**10.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
regression	**20.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	20.0 pts
gradient_descent	**10.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
iterate_gradient	**10.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
compute_betas	**20.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	20.0 pts
predict	**10.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
sgd	**10.0 to >0.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
			Total Points: 100.0