31st October 2017

Peer Review for Anthony Dutcher

I couldn't find your code for the 3rd , 4th or 5th exercise so I have reviewed your code of the Stochastic Gradient Descent optimizer for logistic regression and have found your work to be impeccable. In my capacity, I have made some observations and have the following suggestions and comments about your work,

1. Your code is very well commented, it helped me go through your thought process very easily.
2. I noticed that you have not added any error term to the 'Negative log likelihood' function, though not necessary, this could prevent any unnecessary numerical singularity.
3. In your functions for loss and gradient, you have calculated probability (w) for each of them, an alternative way that could save you some computing power could be calculating the probability once and using it as an input for the functions.
4. I found your code to be highly reusable with the functions dependent on the parameters of the input dataset. This has helped me improve the reusability of my codes.
5. You have pre-processed the dataset in functions you defined for gradient descent and the stochastic gradient descent, another way of doing could be pre-processing the data once and then using it for both the functions.
6. Instead of defining two different functions, you can also define a function for gradient descent that takes batch size as input. For doing this you would also need a feeding function that selects a batch from the data according to the specified batch size.
   Here is a python code for the same:

```
def
GD(X,Y,batch_size,step_size):
                                x,y = feed(X,Y,batch_size)
                                global B
                                w =  activation_sigmoid(logits(B,x))
                                P,G   = penalty_diff_sum(B)
                                loss = neg_log_likely(y,w)
                                B += -(step_size *gradient(x,y,w))
                                return loss


def
feed(X,Y,batch_size):
                            assert len(X)%batch_size == 0 ##to make sure perfect allocation
                        of data
                            global cursor
                            x_train = X[cursor:cursor+batch_size]
                            y_train = Y[cursor:cursor+batch_size]
                            if cursor == len(X)  :
                                cursor = 0
                            else :
```

```
            cursor += batch_size
    return x_train, y_train
```

7. Initializing Beta with all zeroes can cause slower start, using random values from a truncated normal or standard normal could prove to be better.

8. In the predict function, it would be a good Idea to add probability threshold as an input parameter.

Reviewing your work was very helpful. The comparison plots are revealing and informative, and provided me with some insights that I have missed to appreciate in my work. The above suggestions are highly dependent on my personal understanding and I hope that you find them helpful.

Ashutosh Singh