module-08-programming-only

March 24, 2019

1 Module 8 - Programming Assignment

1.1 Directions

There are general instructions on Blackboard and in the Syllabus for Programming Assignments. This Notebook also has instructions specific to this assignment. Read all the instructions carefully and make sure you understand them. Please ask questions on the discussion boards or email me at `EN605.645@gmail.com` if you do not understand something.

You must follow the directions *exactly* or you will get a 0 on the assignment.

You must submit your assignment as `<jhed_id>.py`. You do not need to submit the data files.

2 Linear and Logistic Regression

For this assignment, you are going to implement both linear and logistic regression. For a starting point, you can refer to **module-8-pseudocode.pdf**.

Remember, the only actual difference between the two algorithms is f() and the error function, you should consider having a common general implementation and have the specific functions delegate using higher ordered functions.

In order to complete this assignment, you will need some data. There are two files accompanying this assignment, linear_regression.csv and logistic_regression.csv

The files are CSV files (comma separated values) and have the same basic layout:

x1, x2, y with no headers.

You can google how to read a CSV file. You should assume that the last value in each row will always be the value you need to predict (y) but that there may be any number of explanatory (feature) values (xs). This means your algorithm should work with data of any size...don't hard code anything. You need to add the x0 = 1 (bias) term in both the learn and apply functions.

In the lecture, I mentioned that you usually should mean normalize your data but you don't need to do that in this case because the data is already mean normalized.

I should mention that gradient descent is not the usual approach to linear | logistic regression because the error function actually has an *exact* solution. However, in the case of large data sets, the exact solution often fails and in any case, the use of gradient descent will prepare you for neural networks next week.

if you want to include numpy and implement a vectorized version, please go ahead. You might consider a non-vectorized version first.

You may NOT use Pandas or any library (including the linear model function of numpy.

There are three basic functions:

1. `read_data` will take a filename and return the data as a List of Lists.
2. a "learn" function that takes data in the format returned by `read_data` and returns a List of weights (thetas), the "model".
3. an "apply" function that takes a List of weights (the model), and then just the xs for a problem and returns a value.

The learn functions take an optional parameter, debug, that defaults to False. If set to True, it will print out the error for each iteration so that you can make sure the error is is decreasing. In general, if the error starts to increase again the learning rate is too large. You can add an adaptive learning rate to your function that decreases the learning rate whenever the error increases.

In general, the error rate should not increase. An increasing error rate is the sign of an incorrect ("not an 'A'") program.

2.1 Learning the Model

```
In [ ]: def read_data( filename):
            pass
```

```
In [ ]: def learn_linear_regression( data, verbose=False):
            pass
```

```
In [ ]: def learn_logistic_regression( data, verbose=False):
            pass
```

2.2 Applying the Model

Since the functions above learn a model, you need separate functions to apply the model to new data. If the data is training or test data, you need to ignore the y value when making the prediction (see the print loop below) and just use the $x's$. If this is data you've never seen before, you don't know what y is.

Remember to add x_0 = 1. This means that model will always be one longer than xs because model will contain θ_0.

```
In [ ]: def apply_linear_regression( model, xs):
            pass
```

```
In [ ]: def apply_logistic_regression( model, xs):
            pass
```

Your code, when done, should properly execute the code below:

```
In [ ]: data = read_data( "linear_regression.csv")
        linear_regression_model = learn_linear_regression( data, verbose=True)
        print( "model:", linear_regression_model)
        for point in data[ 0:10]:
            print( point[-1], apply_linear_regression( linear_regression_model, point[:-1]))
```

```
In [ ]: data = read_data( "logistic_regression.csv")
        logistic_regression_model = learn_logistic_regression( data, verbose=True)
        print( "model:", logistic_regression_model)
        for point in data[ 0:10]:
            print( point[-1], apply_logistic_regression( logistic_regression_model, point[:-1])
```