CS 422/622 Project 2

Due 10/20 11:59pm

Logistics: You must implement everything stated in this project description that is marked with an **implement** tag. Whenever you see the **write-up** tag, that is something that must be addressed in the project README.txt. Graduate students are required to implement everything including items tagged with **622**. Students in 422 do not need to complete these extra elements. **You cannot import any packages unless specifically noted by the instructor.** In this project, you may import the numpy and math packages. You are welcome to ask about particular packages as they come up. The idea is to implement everything from scratch.

Deliverables: Each student should submit a single ZIP file, containing their project code (*.py files) and your writeup (README.txt). You should create a folder called Project2 (exactly like this with the capital P) and put all your code and your writeup in this folder. Then zip this folder up. That way when I extract your folder I can run my tests from outside the directory without having to change anything. Your zip file should be named lastname_firstname_project2.zip. Your code should run without errors in a linux environment. If your code does not run for a particular problem, you will lose 50% on that problem. You should submit only one py file for each problem. If your file does not match the filename provided exactly, you will lose 50% on that problem.

Grading: I have provided you with a test script (test_script.py) you can use to test out your functions. I will be using a similar test script, so if your code works with this script it should work with mine! The output of the test script should look something like this if you have implemented everything correctly. **Each student must work independently. You are to submit your own original work.**

1 Perceptron (40 points)

File name: **perceptron.py**

Implement a function in python:

```
perceptron_train(X,Y)
```

that takes training data as input and outputs the weights `w`, and the bias `b` of the perceptron. Your function should handle any real-valued features, with feature vectors in any dimension, and binary labels. **Write-Up**: describe your implementation.

Implement a function in python:

```
perceptron_test(X_test, Y_test, w, b)
```

that takes testing data, the perceptron weights and bias as input and returns the accuracy on the testing data. **Write-Up**: describe your implementation.

You should get the following output for the perceptron when you run the test script.

```
Perceptron: 1.0
Perceptron: 0.42857142857142855
```

2 Gradient Descent (20 points)

File name: **gradient_descent.py** **Implement** a function in python: $gradient\_descent(\nabla f, x_{init}, \eta)$ that takes the gradient of a function f as input, the starting x_{init} and the learning rate η. The output of $gradient\_descent$ is the value of $\mathbf{x}$ that minimizes ∇f. ∇f will be in the form of a function, so that you can calculate the gradient at a particular point. That is ∇f is a function with one input $\mathbf{x}$ and it outputs the value of the gradient at that point. If we are working with 1D variables, then $\mathbf{x}= (x_1)$. If $\mathbf{x}$ is 2D then $\mathbf{x}= (x_1, x_2)$ and so on. $\mathbf{x}$ should be a 1D numpy array. **Write-Up**: describe your implementation.

If you test your function using $f(x) = x^2$, $x_{init} = 5$, and $\eta = 0.1$. $\nabla f = 2x$. You should get a final x of $4.56719262e-05$ or something very small like that. It should take 52 steps. Note, I am not checking for gradient equal to 0. I am checking that the gradient is less than some very small value, 0.0001. You may use whatever check you like. Please report it in your write up.

For testing purposes: I will test your program with various ∇f.

You should get the following output for the gradient descent when you run the test script.

```
Gradient Descent: [4.56719262e-05]
Gradient Descent: [2.16433186e-55 5.77016976e-03]
```

3 Linear Classifier (40 points)

File name: **linear_classifier.py**

Implement a function in python:

`linear_train(X,Y,dLdw,dLdb,eta)`

that takes training data as input as well as the partial derivatives of the loss function and η, and outputs the weights `w`, and the bias `b` of the trained linear classifier. Your function should handle any real-valued features, with feature vectors in any dimension, and binary labels. **Write-Up**: describe your implementation.

Implement a function in python:

`linear_test(X_test, Y_test, w, b)`

that takes testing data, the linear classifier weights and bias as input and returns the accuracy on the testing data. **Write-Up**: describe your implementation.

For testing purposes: I will test your program with various ∇L.

You should get the following output for the perceptron when you run the test script.

```
Linear Classifier: 0.42857142857142855
```