CSE 326/426 (Fall 2020) Project 1

Due on 11:55pm Eastern Time, Sep 16, 2020

Goal: Implement logistic regression for binary classification.

Instruction: Decompress the file project_1.zip and find the following file structure

```
|-- data
|   |-- SGD_outcome.pkl
|   |-- X.txt
|   |-- batch_outcome.pkl
|   |-- newton_outcome.pkl
|   `-- y.txt
|-- plot_training_log.ipynb
|-- problem1.py
|-- problem2.py
|-- problem3.py
|-- problem4.py
|-- problem5.py
|-- test1.py
|-- test2.py
|-- test3.py
`-- test4.py
```

You will implement functions to train a logistic regression model that predicts binary class distribution for any input feature vectors/matrices, using gradient descent and Newton method[1] for model optimization. Below is a summary of the tasks and more detailed instructions are given in the comments of the functions.

Algorithm implementation: in the files problem∗.py ($* = 1, \ldots, 5$), there are functions for you to fill out with your codes. Look for blocks look like

```
#########################################
## INSERT YOUR CODE HERE
#########################################
```

Place your implementation under those blocks.

- Step 1. Read and agree the rules stated in problem1.py, which is for you to become familiar with the grading system.
- Step 2. Read in and process the input data that are in the files

  ```
  data/X.txt
  data/y.txt
  ```

[1] Required for graduates

which will be spitted into training/test datasets in problem2.py.

- Step 3. In problem3.py, compute the gradient of negative log-likelihood loss function with respect to the model parameter $\boldsymbol{\theta}$ on a single training example $(\mathbf{x}^{(i)}, y^{(i)})$. This will be used in stochastic gradient descent (SGD) in problem5.py.
- Step 4. In problem4.py, implement a vectorized gradient computation on the entire batch of m training examples $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$. Any form of loops in Python, including but not limited to `for` or `while`, are prohibited. Instead, you should exploit the matrix and vector manipulation functionalities in numpy to compute the gradient. If you're not familar with numpy matrix/vector manipulations, please go back to HW0 and refer to the Stanford tutorial on numpy. This will be used in the batch gradient descent in problem5.py.
- Step 5. In problem5.py, implement the stochastic and batch gradient descent algorithms (and Newton method, required for only graduates) training algorithms to train the model. The training log will be output to the three files, respectively,

  ```
  data/SGD_outcome.pkl
  data/batch_outcome.pkl
  data/Newton_outcome.pkl
  ```

  from which the jupyter notebook

  ```
  plot_training_log.ipynb
  ```

  will plot training and test losses over the course of training. It also plots the norm of parameters to verify that your optimization is indeed convergent (should plateau at the end).

Algorithm testing: files test$*$.py ($* = 1, \ldots, 4$) will test the correctness of your implementations in the corresponding problem$*$.py files. For example, after you implement problem1.py file, run

```
nosetests -v test1
```

to test the correctness of your implemented functions in problem1.py (not graded). If you use Anaconda (highly recommended), there should not be any packages you need to install. Otherwise, you will need to install nosestests for such unit test.

Simple test data are provide in the test files and the success in passing these tests does NOT guarantee that your program will train the model successfully on the real dataset.

Grading: This project has 100 points in total. The number of points for each functions in problem1.py to problem4.py are printed when you run nosetests. problem5.py is graded based on the output training log, plotted using the jupyter notebook "plot_training_log.ipynb". The project counts towards 10% of the final grade (40% for all projects).

Submitting: There is no hand-written report required, and your submission should include the ONLY file

```
<your LIN>P1.zip
```

which, when decompressed, contains the .py files shown in the tree above, with functions in problem$*$.py filled out. Submit the zip file to Coursesite.