CIS 472/572, Winter 2019

Homework 4 (Programming): Linear Models

DUE DATES: Submit on Canvas by Sunday, May 19th at 11:00pm.

1 Overview

In this assignment, you will implement the perceptron algorithm and logistic regression. The file format is the same as in the previous projects: binary attributes and a binary class label represented as comma-separated values.

Please use the provided templates `perceptron.py` and `lr.py` for the perceptron and logistic regression algorithms, respectively. Fill in your code for the training and prediction functions, `train_perceptron`, `predict_perceptron`, `train_lr`, and `predict_lr`. You may add any additional functions you find helpful.

Your code must be your own. **Undergraduates may complete the assignment in teams of 2.** Graduates must complete the assignment alone.

Once you complete the template code, it can be run from the command line with the following arguments:

```
python perceptron.py <train> <test> <model>
python logistic.py <train> <test> <eta> <lambda> <model>
```

Where `train` is the name of a file containing training data, `test` contains test data to be labeled and `model` is the filename where you will save the resulting linear model. For logistic regression, `eta` is the learning rate and `lambda` is the scale of an L2 regularizer.

2 Model File Format

For saving model files, use the following format:

```
0.10976
foo -1.2857
bar  0.4811
```

where the first line contains the bias, and each line that follows lists one of the attributes and its corresponding weight.

2.1 Perceptron

`train_perceptron` should run the perceptron algorithm until convergence. This means running until all training examples are classified correctly. Your implementation should also halt after 100 passes through the training data, if it has not yet converged. You do not need to shuffle the example order for this question.

`predict_perceptron` should return the activation $wx+b$ for a given example x with a given model (w, b).

2.1.1 Logistic Regression

`train_lr` should run the logistic regression learning algorithm for 100 iterations or until convergence. Each iteration should perform one step of batch gradient descent with the given learning rate – do not use stochastic gradient for this assignment. You may assume the algorithm has converged when the gradient is small (e.g., its magnitude is less than 0.0001). On many datasets, logistic regression will use the full 100 iterations without converging.

Use the $\frac{\lambda}{2}\sum_i w_i^2$ as the regularizer, where λ is the strength of the L2 regularizer. The gradient of the regularizer with respect to each w_j is λw_j. When implementing gradient descent, just add the gradient of the regularizer to the gradient of the logistic loss.

`predict_lr` should return the probability that $y = +1$ for a given example x according to a logistic regression model with the given parameters (w, b).

3 Extra Credit

You should understand the mechanism of neural network: forward propagation, back-propagation, and gradient updating by now. However, in practice, we don't need to implement them, because popular machine learning libraries support a feature called auto-gradient, which handles gradient calculation and weights updating for you. Your main job is to build the model. If you have the time, you can work on this MNIST hand written digits recognition problem to get extra credits.

First, read through a pyTorch tutorial on MNIST dataset, which is provied as `pytorch-on-mnist.ipynb`. Then, you will build following 7 models to test on MNIST dataset, and report your accuracy and loss on test set:

- Multi-layer perceptron with one layer of 16 hidden sigmoid units (no dropout).
- Multi-layer perceptron with one layer of 128 hidden sigmoid units (no dropout).
- Multi-layer perceptron with one layer of 128 hidden rectified linear units (ReLU) (no dropout)
- Multi-layer perceptron with one layer of 128 hidden ReLUs (50% dropout).
- One 2D convolutional ReLU layer (as in the tutorial), one layer of 128 ReLUs (50% dropout).
- One 2D convolutional ReLU layer (as in the tutorial), one max pooling layer (as in the tutorial) (25% dropout), one layer of 128 ReLUs (50% dropout).
- Two 2D convolutional ReLU layers (as in the tutorial), one max pooling layer (as in the tutorial) (25% dropout), one layer of 128 ReLUs (50% dropout).

Note: It's recommended that work on Google colaboratory instead of setting up pyTorch environment on your machine. Submit your final .ipynb file on Canvas, which includes the 7 models along with their results.