University of Pittsburgh
CS 2710 Foundations of Artificial Intelligence — Handout 11
Professor Milos Hauskrecht — November 12, 2020

Problem assignment 11
Due: November 19, 2020

Machine Learning

Problem 1. Classification of handwritten digits with a logistic regression model

In this problem we use a logistic regression model to learn and to discriminate between the two digits (3 and 5) that can appear on the input. The input consists of a two dimensional array of 1s and 0s representing the grid of pixels and their colors (black and white), e.g.

```
# ######
      ##
     ##
  ######
      ###
#      ##
# ##   ##
  #######
```

Our objective is to train the model to classify digits 3 and 5 from the set of handwritten digit patterns and their labels. The data are divided into two sets:

- training set used for training the model (files *digit_x.txt*,*digit_y.txt*) ;
- testing set used for testing the performance of the learned model (files *digit_x_test.txt* and *digit_y_test.txt*).

The data are available on the class web page. The data in files are organized in rows. In file *digit_x.txt* rows represent the binary description of either a digit 3 or 5. File *digit_y.txt* reflects its corresponding label, 1 for 3 and 0 for 5. Test files *digit_x_test.txt* and *digit_y_test.txt* have the same structure. There are 100 sample digits in the training, and 400 in the testing files respectively.

To simplify the data load, you are given the *LogReg_data_load.py* file that loads the data into Numpy matrix form (or pandas matrix form in the auxiliary step).

Part a. In the logistic regression model we express the distribution of outputs $p(y = 1|\mathbf{x}, \mathbf{w})$ as:

$$p(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp\left(-\left[w_0 + \sum_{j=1}^{k} w_j x_j\right]\right)},$$

where exp stands for the exponential function, and $\mathbf{w}$ is a vector of weights $w_0, w_1, \cdots, w_{64}$, such that w_0 corresponds to the bias weight and $w_1, \cdots, w_{64}$ to the weights for digit pixels. x_j is the j-th component of the input vector $\mathbf{x}$.

Implement a python function $LogReg\_probability(w, x)$ that takes the set of weights $\mathbf{w}$ and the input vector $\mathbf{x}$, and calculates and returns $p(y = 1|\mathbf{x}, \mathbf{w})$. Include the function in the *LogReg.py* file.

Part b. When learning the logistic regresion model we want to maximize the likelihood of the predictions. The optimization of the likelihood can be performed using the online algorithm with gradient-based updates. The $(i+1)$th on-line update of a weight w_j with a data sample $< \mathbf{x}, y >$, is

$$w_j^{(i+1)} = w_j^{(i)} + \alpha(i + 1) \cdot \left[y - p(y = 1|\mathbf{x}, \mathbf{w}^{(i)})\right] \cdot x_j,$$

where $\alpha(i + 1)$ is the learning rate that changes with the update step, $x^{(j)}$ is the jth component of the input vector $\mathbf{x}$.

Implement a python function: $LogReg\_online\_GD(x\_train, y\_train, num\_iterations)$ that takes training data matrices (inputs x and outputs y) and returns the weights of the parameters of the logistic regression model learned through the online gradient descend process. Your function should:

- start from zero weights (all weights $\mathbf{w}$ set to 0 at the beginning).
- use the annealed learning rate $\alpha(i) = \frac{1}{2\sqrt{i}}$ where i indexes the ith update step.
- in every step, pick randomly a training instance (digit and its corresponding label) from all digits in the training data.

After implementing the function include it in the *LogReg.py* file. Also use the function and run it for 1500 iteration steps on the training data. Report the weights found.

Part c. The model (weights) learned in part b can be used to classify digits. The output label $\{0, 1\}$ for an input $\mathbf{x}$ can be obtained using the following simple rule:
If $p(y = 1|\mathbf{x}, \mathbf{w} \geq 0.5$ then output 1
else output 0.

Write function $LogReg\_classify(w, x)$ that takes model parameters w and input x and assigns a class label to x as predicted by the model. After implementing the function include it in the *LogReg.py* file.

Use the above function and function *LogReg_probability* from Part a. together with model weights from Part b. to calculate and report (1) p(y=1—**x, w** (2) class label for the first 30 digits in the testing dataset. Compare the predictions to the labels given to them in the y test matrix. Discuss the observed results.

Part d. Next are interested in analyzing the quality of the learned logistic regression model on all test data. To test the model quality we use average misclassification error. The average misclassification error for the dataset with N samples is defined as:

$$Error_{AMC} = \frac{\text{\# of misclassified digits}}{N}.$$

Write a python function $mis\_error(predictedlabels, truelabels)$ to compute the average misclassification error of the model and include it in the *LogReg.py* file.

After implementing it use the model weights from Part b. to calculate the misclassfication errors on both the training and testing datasets. Compare and analyze the results in the report. Which misclassification error is higher? Training or testing? Can you explain it?

Part e. Submit *LogReg.py* implementing all above defined functions. After that write and submit *LogReg_Run.py* that uses the above functions to (1) learn the weights based on the training data set (as described above) and 1500 iterations and (2) tests the quality of the resulting model according to part d. Run/iterate the steps 20 times and output (1) the weights learned and (2) average training and testing misclassification error. Report average training and testing misclassification error over these runs.

Remark: Although the resulting models can differ because of the random choice of the training examples you should be able to see models with the average misclassification error for the testing data below 0.15 most of the time.

Part f. Optional (Extra credit). You may want to experiment a bit with the number of iterations and learning rate schedules. For example, try to learn using only 100, 200, 500 updates before returning the weights. In terms of learning schedule you can try e.g. $\alpha(i) = \frac{1}{i}$. Report results of your experiments and any hypothesis or conjectures about the performances.