Assignment 3 Report

Xiaopei Zhang   A20302816

I. Problem Statement

This assignment requires to implement various techniques in discriminative learning from scratch, evaluate the performance of different algorithms, and figure out how to predict class which each given test case belongs to. The first two sections focus on logistic regression. It includes 2-class dataset and k-class dataset. Python logistic regression and confusion matrix are used to compare and evaluate performance. The last section is mainly about neural network for 3-class classifications with momentum. Confusion matrices are calculated, and Python Multiple-Layer Perceptron is used for comparison with the 3-class classifier.

II. Proposed Solution

In the first sections, I use the following functions and steps to classify 2-class dataset.

1. Sigmoid function.

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

2. Gradient descent.

$$\theta_j \leftarrow \theta_{j-1} - \eta \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right) \cdot x^i$$

Classify by computing Sigmoid(x) = h(x). If h(x) is closer to 1, x belongs to class 1; if it is closer to 0, x belongs to class 2.

In the second section, I use one vs the rest method and also use Sigmoid and gradient descent to get thetas for different classes. If the highest h(x) is resulted from theta for class c, then x belongs to class c.

Written up steps for neural network with 2 classes using maximum likelihood:

$$l_j(\theta) = \sum_{i=1}^{m}[y^{(i)}\log(P(y=j\,|\,x^{(i)})) + (1-y^{(i)})\log(1-P(y=j\,|\,x^{(i)}))]$$

$$\frac{\partial l_j(\theta)}{\partial P(y=j\,|\,x^{(i)})} = \sum_{i=1}^{m} y^{(i)}\cdot\frac{1}{P(y=j\,|\,x^{(i)})} - \sum_{i=1}^{m}(1-y^{(i)})\cdot\frac{1}{1-P(y=j\,|\,x^{(i)})}$$

$$\frac{\partial P(y=j\,|\,x^{(i)})}{\partial v_j} = P(y=j\,|\,x^{(i)})(1-P(y=j\,|\,x^{(i)}))z^{(i)}$$

$$\frac{\partial l_j(\theta)}{\partial v_j} = \frac{\partial l_j(\theta)}{\partial P(y=j\,|\,x^{(i)})}\frac{\partial P(y=j\,|\,x^{(i)})}{\partial v_j} = \sum_{i=1}^{m}(P(y=j\,|\,x^{(i)}) - y_j^{(i)})z^{(i)}$$

$$\frac{\partial l_j(\theta)}{\partial P(y=l\,|\,x^{(i)})}\frac{\partial P(y=l\,|\,x^{(i)})}{\partial z_j}\frac{\partial z_j}{\partial w_j} = \sum_{i=1}^{m}(\sum_{l=1}^{k}(P(y=l\,|\,x^{(i)}) - y_l^{(i)})v_{lj})z_j^{(i)}(1-z_j^{(i)})x^{(i)}$$

In the last section, I use Softmax and the following function for neural network 3-class gradient descent.

1. Softmax.

$$h_{\theta j}(x) = \frac{\exp(\theta_j^T x)}{\sum_{i=1}^{k}\text{xp}(\theta_i^T x)}$$

2. Gradient descent.

$$w_j^m \leftarrow w_{j-1}^m - \eta\sum_{i=1}^{m}(\hat{y}^i - y^i)\cdot v_m\cdot z_m^i\cdot(1-z_m^i)\cdot x^i$$

3. Momentum.

$$w_j^{t+1} \leftarrow w_j^t - \eta\frac{\partial l}{\partial w_j} + \beta(w_j^t - w_j^{t-1})$$

III. Implementation

1. Program Design

My implementations of all the functions are highly modular. According to the sequence of my implementation, I explained most related functions below.

sigmoid: take in theta and z, return the sigmoid value of the dot product of theta and z

softmax: take in theta and z, return the softmax value of the dot product of theta and z

gradientDescent: take in initial theta value, training z, training y, learning rate and iteration number, use gradient descent method, and return calculated theta

logisticPredictY: take in theta and testing z, return the class z belongs to.

logisticPredictYMultiple: take in all the probabilities of one test case, return the class this test case belongs to.

sigmoidDerivative: take in a value, return the derivative of this value

neuralNetwork3Classes: take in training x, training y, the number of hidden dimensions and learning rate, and return theta0 to get the hidden layer and theta0 to get y

neuralPredictY3: take in testing x, theta0 to get the hidden layer and theta0 to get y, and return probability

neuralClassifier: take in 3 probability values, and return the class by the highest probability

In the main function, "#" printout separates the four sections. In first two section, I create global confusion matrices for results from my own functions. In the last section, I use local confusion matrices within each pair of training dataset and testing dataset. Under 10 fold cross validation, I perform logistic regression on iris_reorganized_2class.data and iris_reorganized.data (3-class), neural network on ex4data1.mat (digital data from Andrew Ng's machine learning class). In my script, for 3-class neural network I extract the first three-class data (label 1, label 2 and label 3). Correspondingly, I perform Python made functions Logistic Regression, and Multiple-Layer Perceptron for comparison of confusion matrices and evaluation.

2. Problems Encountered

For logistic regression part, the classification was initially not good enough. After mapping to higher dimensional space, it becomes very good. For neural network part, the performance of my function is not as good as Python made function.

3. Instructions

My script is written and tested under Python 2.7. My sklearn version is 0.17.1, and scikit-neuralnetwork is 0.6.1. Please keep my script and data under one directory before execution. If you would like to change the number of hidden dimensions or learning rates, please do it to line 293 - 295, line 305 and line 307.

IV. Results and Discussion

1. Results

2-class Logistic Regression (both linear and nonlinear)

```
My logistic regression:
[[20   0]
 [ 0 80]]
Python logistic regression:
[[20   0]
 [ 0 80]]
My logistic regression nonlinear:
[[20   0]
 [ 0 80]]
Python logistic regression nonlinear:
[[20   0]
 [ 0 80]]
```

3-class Logistic Regression

```
My k-class logistic regression nonlinear:
[[100    0    0]
```

```
 [   0   39    2]
 [   0    1    8]]
```
Python k-class logistic regression nonlinear:
```
[[100    0    0]
 [   0   41    0]
 [   0    1    8]]
```

3-class Neural Network

My 3-class neural network:
```
[[90  4 14]
 [ 2 19 17]
 [ 0  0  4]]
```
Python 3-class neural network:
```
[[108    0    0]
 [   0   37    1]
 [   0    0    4]]
```

2. Demonstration of Correctness

My implementations of Logistic Regression and Neural Network are correct since my confusion matrices are similar with confusion matrices derived from Python made functions. Also, my implementations are based on the given functions in section II, so they are theoretically correct.

3. Performance Evaluation

(1) Strength. All of my implementations are theoretically derived, and my logistic regression has almost the same high accuracy as Python made function. My neural network is much faster than Python made function, thought I guess that it might be because Python has more complex processes to improve correctness.

(2) Weakness. My implementation of neural network has a lower accuracy than Python made function, and its accuracy hinges highly on my settings of learning rate and hidden dimension.

4. Effect of Various Parameters on Results

I change the learning rate and the number of hidden dimensions to see how neural network performs. I found that for both parameters there is an optimal value. If my setting is below or above the optimal value, the accuracy of the output get lower. The optimal value I found for learning rate is around 0.2, and the optimal value I found for hidden dimensions is around 20.

References

[1]http://archive.ics.uci.edu/ml/datasets/Iris
[2]https://github.com/justinj656/ml-Andrew-Ng/blob/master/mlclass-ex4/ex4data1.mat
[3]https://pypi.python.org/pypi/scikit-neuralnetwork