cs584 Assignment 3: Report

Xin Su
Department of Computer Science
Illinois Institute of Technology

March 31, 2016

Abstract

This report states the analysis of the Assignment 3's results. It resolves 2 major problems one by one. First I implement a Logistic Regression models to fit the datasets of 2-class, 2-class with non-linear feature combinations and k-class distribution. Second I implement an Multipayer Perceptron model and fit it to the datasets with 2-layer feedforward and 3-class. I evaluate the results for each model using confusion matrix, and accuracy. I also derive the backpropagation update equations for the weights of the output by minimizing the error function.

1. Problem statement

I explored the following problems:

(1) Implement the Logistic Regression algorithm for 2-class discrimination.
(2) Use non-linear combinations of inputs to increase the capacity of the classifier.
(3) Implement the Logistic Regression algorithm for k-class discrimination.
(4) Derive the backpropagation update equations for the weights of the output by minimizing the error function.a 2-layer feedforward MLP for 3-class classification.
(5) Implement Multipayer Perceptron model and fit it to the datasets with 2-layer feedforward and 3-class.

2. Proposed solution

- Logistic Regression for k-class discrimination
  - Hypothesis function (sigmoid function)

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

  - Objective function

$$l(\theta) = -\sum_{i=1}^{m} \left(y^{(i)} log(h_\theta(x)) + (1 - y^{(i)}) log(1 - h_\theta(x)) \right)$$

- Gradient descent function

$$\theta_j = \theta_j - \alpha(h_{\theta j}(x) - I(y = j))x^j$$

- Predict function

$$\dot{y} = argmax_j \theta^T X$$

- Multilayer Perceptron with 2-layer k outputs
  - Hypothesis function

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

  - Objective function

$$l(\theta) = -\sum_{i=1}^{m}\sum_{j=1}^{k} I(y^{(i)} = j) log\left(\dot{y}_j^{(i)}\right)$$

  - Gradient descent function

$$V_j = V_j - \alpha \sum_{i=1}^{m} \left(\dot{y}_j^{(i)} - y_j^{(i)}\right)$$

$$W_j = W_j - \alpha \sum_{i=1}^{m}\sum_{l=1}^{k} \left(\dot{y}_l^{(i)} - y_j^{(i)}\right) V_{lj} z_j^{(i)} \left(1 - z_j^{(i)}\right) x^{(i)}$$

  - Predict function

$$\dot{y} = sigmoid(V^T z)$$

$$z_j = sigmoid\left(W_j^T x\right)$$

$$h = \text{argmax}_j \dot{y}$$

- Evaluation function
    - Confusion Matrix:

		Label Classifier		
		C1	**…**	**Ck**
Predict Classifier	**C1**	C11	…	C1k
	…	…	…	…
	Ck	Ck1	…	Ckk

    - accuracy

$$accuracy = \frac{\Sigma C_{ii}}{\Sigma C_{ij}}$$

3. Implementation details

I use Python 2.7 as development language, and use IPython Notebook to write the code and run. It is very convenient for use. A user can view it simply using a browser, and the content will contain all output, including text, data and graphs.

There is an additional requirement to run the scikit-learn code. Please follow the link [4] and [5] in the Reference to install scikit-learn 0.18 dev0 first.

When you want to run the code, you should first install the IPython Notebook (Please refer to the Jupyter official website: http://jupyter.readthedocs.org/en/latest/install.html). After this step, you can open the .ipynb file and run. Just run from the top cell to the bottom. This top-down order is required.

4. Results and discussion

Problem 1

I use iris.data dataset but only choose the second and third class. For the initial value of theta, I use all 0. The result shows that the cross validation validates the model. The accuracy is pretty high and the reason for that is the data in this dataset is really good and express the features uniquely. As the amount in this dataset is small, the running time is small as well.

Problem 1 Result

	cross validation training avg	cross validation test avg
Confusion	[[34. 0.] [4. 42.]] [[42. 0.] [2. 36.]] [[35. 0.] [3. 42.]] [[37. 0.] [3. 40.]] [[37. 0.] [3. 40.]]	[[12. 0.] [0. 8.]] [[6. 0.] [0. 14.]] [[11. 0.] [1. 8.]] [[8. 0.] [2. 10.]] [[9. 0.] [1. 10.]]
accuracy	0.9625	0.96
duration (s)	10.8072888851	

Problem 2

I still use iris.data dataset with two classes. For the initial value of theta, I use all 0. The result shows that the cross validation validates the model, the accuracy is still high, but not as high as previous result. The reason for this may be that when I use non-linear features, this model won't fit the dataset very well as the features are more close to linear relation. For the running time, it runs faster. The reason may be that the additional non-linear features makes the fit process more easier.

Problem 2 Result

	cross validation training avg	cross validation test avg
Confusion	[[32. 1.] [6. 41.]] [[34. 1.] [4. 41.]] [[36. 1.] [6. 37.]] [[43. 19.] [0. 18.]] [[33. 1.] [6. 40.]]	[[11. 0.] [1. 8.]] [[9. 0.] [3. 8.]] [[7. 0.] [1. 12.]] [[7. 7.] [0. 6.]] [[10. 0.] [1. 9.]]
accuracy	0.8875	0.87
duration (s)	0.25905418396	

Problem 3

I still use iris.data dataset and use all classes. For the initial value of theta, I use all 0. The result shows that the cross validation validates the model, and it also shows that the accuracy are also high which means the model fits well. As the running time. it will take more time to run. As the amount of dataset raises 50% compared to when using two class. the time used is also raised about 50%.

Problem 3 Result

	cross validation training avg	cross validation test avg
Confusion	[[43. 0. 0.] [0. 27. 8.] [0. 13. 29.]] [[40. 0. 0.] [0. 23. 4.] [0. 17. 36.]] [[37. 0. 0.] [0. 31. 8.] [0. 12. 32.]] [[39. 0. 0.] [0. 30. 5.] [0. 12. 34.]] [[41. 0. 0.] [0. 9. 0.] [0. 26. 44.]]	[[7. 0. 0.] [0. 3. 1.] [0. 7. 12.]] [[10. 0. 0.] [0. 3. 1.] [0. 7. 9.]] [[13. 0. 0.] [0. 6. 1.] [0. 1. 9.]] [[11. 0. 0.] [0. 8. 6.] [0. 0. 5.]] [[9. 0. 0.] [0. 3. 0.] [0. 12. 6.]]
accuracy	0.825	0.76
duration (s)	16.0240461826	

Problem 4

This is the procedure to derive the backpropagation update equations for the weights of the output by minimizing the error function. This is a two-layer two-class MLP and all elements use sigmoid function.

$$\dot{y} = sigmoid(V^T z)$$
$$z_j = sigmoid(W_j X)$$

we have error function:

$$\mathrm{E}(V, W) = \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \dot{y}^{(i)} \right)^2$$

Then we have:

$$\frac{\partial \mathrm{E}}{\partial V} = \frac{\partial \mathrm{E}}{\partial \dot{y}} \frac{\partial \dot{y}}{\partial V} = \sum_{i=1}^{m} \left(y^{(i)} - \dot{y}^{(i)} \right) \dot{y}^{(i)} \left(1 - \dot{y}^{(i)} \right) z$$

$$\frac{\partial \mathrm{E}}{\partial W_j} = \frac{\partial \mathrm{E}}{\partial \dot{y}} \frac{\partial \dot{y}}{\partial z_j} \frac{\partial z_j}{\partial w_j} = \sum_{i=1}^{m} \left(y^{(i)} - \dot{y}^{(i)} \right) V_j z_j (1 - z_j) x^{(i)}$$

Therefore, the update equation is:

$$V = V - \alpha \sum_{i=1}^{m} \left(y^{(i)} - \dot{y}^{(i)} \right) \dot{y}^{(i)} \left(1 - \dot{y}^{(i)} \right) z$$

$$W_j = W_j - \alpha \sum_{i=1}^{m} \left(y^{(i)} - \dot{y}^{(i)} \right) V_j z_j (1 - z_j) x^{(i)}$$

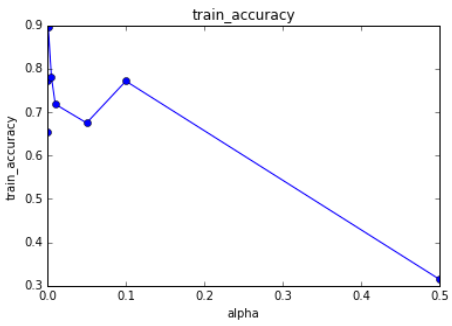
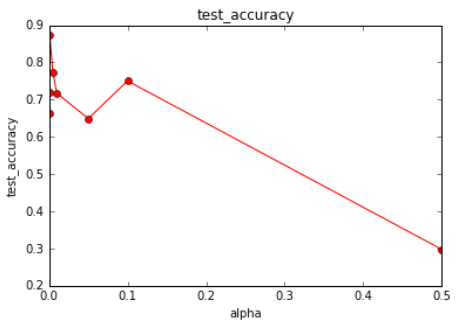
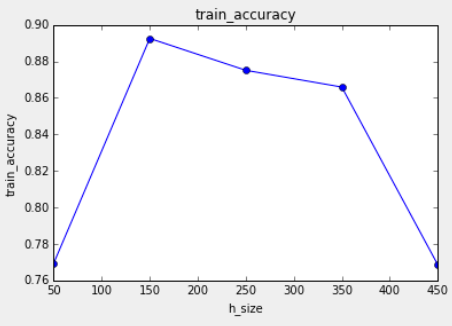
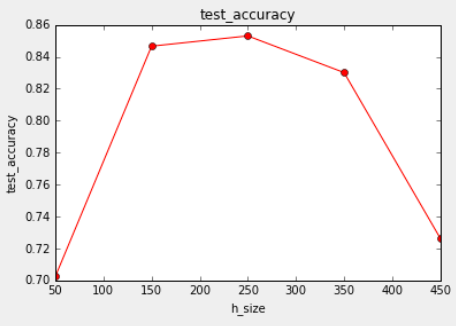
Problem 5

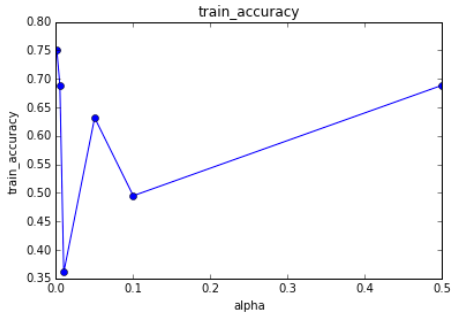
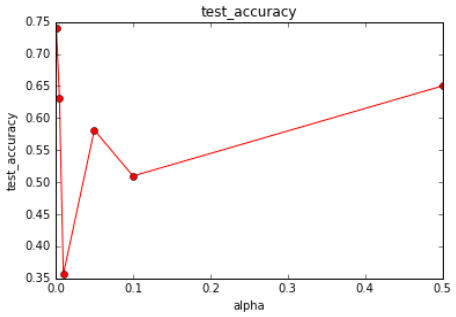
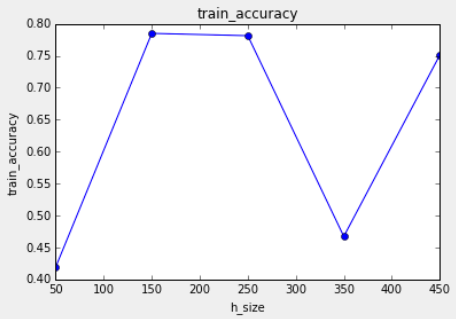
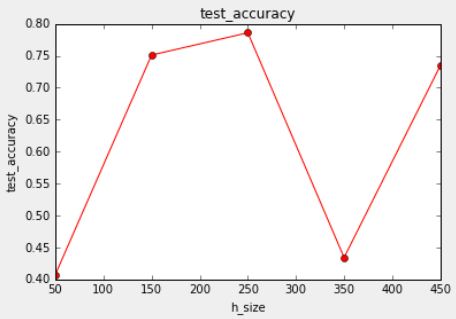
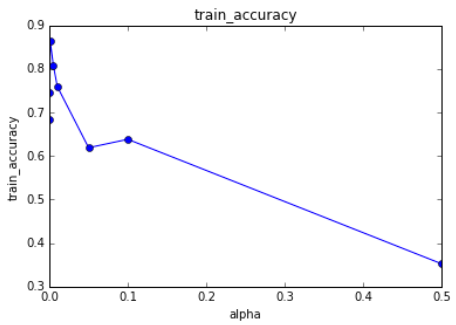
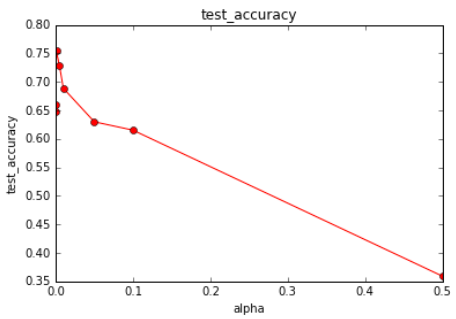
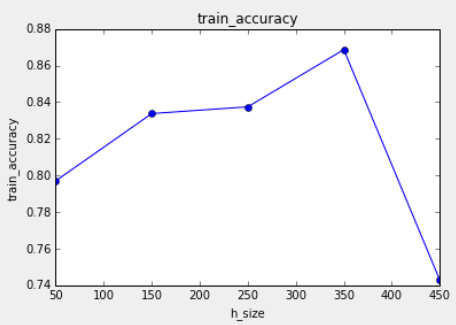
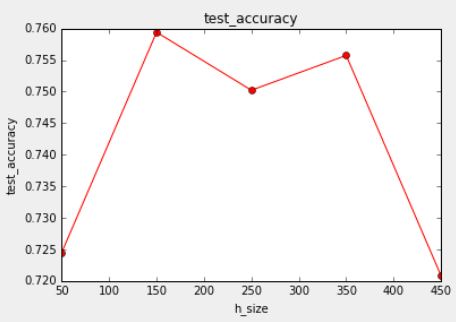
I use MINIST original dataset and use 0, 1, 2, three classes, but shuffle and then truncate it to about 1000 data lines. I use 5 folds to separate the data into training set and test set and I train each data line 100 times to stop as the training process seldom converges.

I tried different initial values for V, W pair, and finally decided to use normal distribution that is close to 0. From the graph as function of accuracy and the number of elements in the hidden layer (Figure 5-1) and function accuracy and of the learning rate parameter (Figure 5-1), we can see these relationships are not exactly linear and this is correct because the neutral network is a non-linear model. From the accuracy, I decided to use 150 elements in the hidden layer and 0.001 as the learning rate, and both the training and test accuracies are around 0.8 to 0.9.

Compared to the scikit-learn algorithm and implementation, my implementation has a lower accuracy, but not too much. However, for the running efficiency, it runs much faster than my code.

Problem 5 - Result

	training accuracy ~ alpha	**test accuracy ~ alpha**
V_1 = 1e-3 * np.random.randn(h + 1, k) **W_1 = 1e-3 * np.random.randn(X.shape[1], h)** **h = 150** **alpha = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]** **max_iter = 100** **n_folds = 5**		
V_1 = 1e-3 * np.random.randn(h + 1, k) **W_1 = 1e-3 * np.random.randn(X.shape[1], h)** **h_size = [50, 150, 250, 350, 450]** **alpha = 0.001** **max_iter = 100** **n_folds = 5**		

	training accuracy ~ alpha	test accuracy ~ alpha
V_1 = np.random.randn(h + 1, k) W_1 = np.random.randn(X.shape[1], h) h = 150 alpha = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5] max_iter = 100 n_folds = 5		
V_1 = np.random.randn(h + 1, k) W_1 = np.random.randn(X.shape[1], h) h_size = [50, 150, 250, 350, 450] alpha = 0.001 max_iter = 100 n_folds = 5		
V_1 = np.ones((h + 1, k)) * (np.sqrt(6) / (y.shape[0] + k)) W_1 = np.ones((X.shape[1], h)) * (np.sqrt(6) / (y.shape[0] + k)) h = 150 alpha = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5] max_iter = 100 n_folds = 5		
V_1 = np.ones((h + 1, k)) * (np.sqrt(6) / (y.shape[0] + k)) W_1 = np.ones((X.shape[1], h)) * (np.sqrt(6) / (y.shape[0] + k)) h_size = [50, 150, 250, 350, 450] alpha = 0.001 max_iter = 100 n_folds = 5		

Problem 5 Accuracy compared to scikit-learn

	My implementation (2000 data lines)	scikit-learn implementation (70000 data lines)
training accuracy	0.897153351699	0.994250

	My implementation (2000 data lines)	scikit-learn implementation (70000 data lines)
test accuracy	0.87419651056	0.976100
duration (s)	70.6242780685	8.38747692108

5. References

[1] https://www.coursera.org/learn/machine-learning
[2] http://www.numpy.org/
[3] https://www.python.org/
[4] http://stackoverflow.com/questions/33568244/upgrade-to-dev-version-of-scikit-learn-on-anaconda
[5] https://github.com/scikit-learn/scikit-learn/issues/3606
[6] http://scikit-learn.org/