Project 2
Comparison of Linear Regression, Logistical Regression and Neural Networks

Muhammad A. Masood
Department of Computer Science
University at Buffalo
State University of New York
UB person# 50291164
mmasood@buffalo.edu

Abstract

The purpose of the project 2, was to differentiate between two images that whether a word is written by the same person or not. For this purpose, three techniques named as linear regression, logistical regression and neural networks were used. For linear regression, noise and basis objective functions were defined and regularization was used to contain the overfitting problem. The closed-form solution and gradient solution were found and the root mean square error was calculated for the models. The same was done for logistic regression and then a neural network model was generated to solve the same problem. Finally, the hyperparameters were changed to different settings and the accuracy of different models was compared.

1 Problem Statement

The datasets consisted of two parts, i.e., human observable datasets and computer-generated datasets. Human observable datasets consisted of nine features and then they were processed into two ways. They were concatenated and in the other way, they were subtracted to get two sets of training data. Similar procedure was followed for computer-generated datasets. So, in effect, there were 4 settings of data that were generated.

2 Experiment

In the experiment, the problem was first implemented with closed-form solution and then with stochastic gradient descent and evaluated with root mean square (rms) error. And then the same was done for logistical regression and neural network methods. Accuracy was used as a measure of performance in the case of logistical regression and neural networks.

2.1 Linear Regression Model

Linear regression function for the model was:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \phi(\boldsymbol{x})$$

w is weight vector in and ϕ is the basis function. The first element of w, w_0 is the bias in this case. The function of the basis function is to map vectors to a scalar value. The basis function used for this model was radial basis function with the equation:

$$\phi_j(\boldsymbol{x}) = e^{(-\frac{1}{2}(\boldsymbol{x}-\mu_j)^T \Sigma_j^{-1}(\boldsymbol{x}-\mu_j))}$$

μ_j is the center of basis function and Σ_j^{-1}defines the spatial spread.

2.2 Regularization

Regularization was added to the error function to reduce the problem of overfitting and for this purpose a weight decay regularizer was used. The equation is as follows:

$$E(w) = E_D(\boldsymbol{w}) + \lambda E_w(\boldsymbol{w})$$

The equation for weight decay regularizer

$$E_w(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$

The goal was to minimize E(w) by selecting appropriate value of **w.**

2.2 Closed-form solution

The closed form solution with a regularizer can be found by using the following equation:

$$\boldsymbol{w}^* = \left(\lambda \boldsymbol{I} + \boldsymbol{\phi}^T\boldsymbol{\phi}\right)^{-1}\boldsymbol{\phi}^T\boldsymbol{t}$$

2.3 Gradient Descent Solution

Stochastic gradient descent takes the derivative of a weight vector and multiplies it with a learning rate. The learning rate can be changed to manage the convergence time of the model. The new weights are then updated in accordance with the following equation:

$$w^{(\tau+1)} = w^\tau + \Delta w^\tau$$

And then we have:

$$\nabla E_D = -\left(t_n - w^{\tau T}\phi(x_n)\right)\phi(x_n)$$

$$\nabla E_w = w^\tau$$

2.4 Evaluation

The solutions were then evaluated on the basis of root mean square (rms) error, which is defined as:

$$E_{RMS} = \sqrt[2]{\frac{2E(w^*)}{N_v}}$$

2.5 Logistic Regression

Logistic regression function for the model was:

$$y(x, w) = \sigma(w^{T}\phi(x))$$

w is the weight vector in this equation and ϕ is the basis function. Sigmoid acts as a sort of activation function in the case of logistic regression.

2.6 Neural Network

Keras was used for the implementation of fizzbuzz task for software 2.0 with the help of a neural network. A training set of numbers ranging from 101 to 1000 was used to train the network. The model defined for the neural network was densely connected neural network with one hidden layer where each input is connected to each output by a weight. The model was a simple sequential neural network with a single possible output for each input which is enough for this classification problem. The input layer consisted of 10 nodes to accommodate for all numbers in the training data which was converted into binary to create a vector. The labels were categorized into 4 values in a list which contained the correct labels of each number for the input layer. Placeholder variables were created for the input tensor and the output tensor. The next layer which was the hidden layer of the network had 256 nodes initially. We used the activation function, rectified linear unit (ReLu) to decide which neurons should be activated. The ReLu activation function can be defined by a simple equation of A(x)= max (0, x). The error function or the cost function was defined to calculate the mean of weighted cross-entropy i.e. closeness between two discrete probabilities, of tensors. The RMSprop optimizer was used for the learning algorithm which divided the gradient by a running average. The output layer uses softmax activation which gave the probability distribution of input over the aforementioned 4 labels.

The model was then trained on mini-batches of size 128 and over 5000 epochs. A dropout rate of 0.2 was used to minimize the overfitting. The model was then tested on validation data of numbers from 1 to 100, and accuracy was measured. An early stopping callback was used to stop the training when the loss function could not be further minimized. The hyperparameters were later tweaked to find the better accuracy measures

3 Code

NumPy was used for the linear and logistical regression implantation of the problem and k-mean algorithm was also imported from sklearns module to get the k-means clustering. The dataset was read and was divided in training, testing and validation tests. Training data consisted of 80% of the dataset while testing and validation each received 10%. The closed form solution was then calculated. It consisted of finding the Moore-Penrose pseudo inverse matrix. In the first step, K-means algorithms was used to convert the data into different clusters and the mean of centroids was calculated. This vector of means was then used, along with the training data to create big sigma. The gaussian radial basis function was then calculated by taking the inverse of big sigma and multiplying it with the difference of data matrix and the means vector calculated from the k-means algorithm. This factor was again multiplied with the transpose of the difference. The weights were

then calculated and a regularization term was added to prevent the overfitting of data. Similar, process was done for testing and validation sets. The root mean square error was then calculated to measure the performance of the model.
In the same fashion, the weights were initialized for the case of stochastic gradient descent and the derivative was calculated and then multiplied with a fixed learning rate. This term was then added to the previous weight matrix to get the new weights. The error function was then calculated and the root mean square error was also determined to measure the performance. The same was done for neural network, and accuracy was measured for all four settings.

4 Results

The hyperparameters of the model were changed for both solutions and the value of root mean square was calculated to see which value gave better results. For closed form solution, clusters of 10, 50, 100 and 200 were chosen. As a general observation, the ERMS value decreased on testing data with increase in number of clusters as shown in table 1 but the trade-off was made in time required for training the data. For increased number of clusters, model took more time to train. The hyperparameters were tuned for all four settings are results have been recorded in the tables below.

5 Figures & Tables

5.1 For First setting

Table 1Model performance with different clusters

Clusters	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
10	0.549	65.3%	72.15%	86.2%
50	0.54	65.5%	72.34%	86.5%
100	0.536	66.6%	73.7%	86.7%
200	0.529	66.62%	73.49%	88.2%

Table 2 model performance with different lambda

Lambda	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0	0.549	65.3%	72.15%	86.2%
0.03	0.549	66.1%	73.89%	87.3%
0.5	0.5	66.17%	73.23%	87.5%
0.9	0.6	66.2%	73.01%	87.28%

Table 3 model performance with different learning rate

Learning Rate	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0.01	0.549	65.3%	72.15%	86.2%
0.05	0.545	65.28%	72.72%	88.3%
0.07	0.56	65.29%	72.43%	87.7%
0.09	0.565	65.45%	72.29%	87.65%

5.2 For Second Setting

Table 4 Model performance with different clusters

Clusters	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
10	0.623	68.1%	75.52%	88.3%
50	0.626	68.43%	75.93%	88.54%
100	0.662	68.56%	75.46%	88.67%
200	0.614	67.96%	74.25%	88.32%

Table 5 model performance with different lambda

Lambda	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0	0.623	68.1%	75.52%	88.3%
0.03	0.549	69.41%	76.84%	88.24%
0.5	0.590	69.37%	76.61%	89.35%
0.9	0.582	69.16%	76.29%	89.72%

Table 6 model performance with different learning rate

Learning Rate	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0.01	0.623	68.1%	75.52%	88.3%
0.05	0.542	70.08%	77.82%	88.25%
0.07	0.557	69.72%	76.31%	88.17%
0.09	0.589	68.54%	75.72%	88.72%

5.3 For Third setting

Table 7 Model performance with different clusters

Clusters	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
10	0.583	66.32%	73.19%	89.2%
50	0.536	68.18%	75.18%	89.53%
100	0.547	68.09%	75.09%	89.12%
200	0.561	67.52%	74.52%	88.81%

Table 8 model performance with different lambda

Lambda	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0	0.583	66.32%	73.19%	89.2%
0.03	0.548	67.23%	74.73%	89.82%
0.5	0.552	68.52%	75.29%	89.24%
0.9	0.561	66.01%	73.91%	89.19%

Table 9 model performance with different learning rate

Learning Rate	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0.01	0.583	66.32%	73.19%	89.2%
0.05	0.545	68.54%	75.81%	89.43%
0.07	0.565	68.23%	75.08%	89.72%
0.09	0.571	67.92%	74.6%	88.91%

5.4 For Fourth Setting

Table 30 Model performance with different clusters

Clusters	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
10	0.614	65.23%	72.5%	84.92%
50	0.582	65.23%	72.91%	86.42%
100	0.586	63.8 %	70.23 %	86.38%
200	0.599	63.52%	70.61%	86.32%

Table 11 model performance with different lambda

Lambda	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0	0.614	65.23%	72.5%	84.92%
0.03	0.527	66.12%	73.75%	85.2%
0.5	0.592	65.16%	72.37%	85.32%
0.9	0.603	65.41%	72.82%	85.17%

Table 12 model performance with different learning rate

Learning Rate	ERMS Linear	Accuracy Linear	Accuracy Logistical	Accuracy Neural
0.01	0.614	65.23%	72.5%	84.92%
0.05	0.590	64.17%	71.65%	85.42%
0.07	0.561	64.66%	71.39%	85.14%
0.09	0.554	64.53%	71.21%	84.96%

6 Conclusion:

Different combinations for the hyperparameters were tried and the best possible root mean square measure was achieved when the clusters were increased to 50 and lambda of 0.03 was chosen for linear regression. Same settings produced the best results for logistical regression. For neural network, the best possible accuracy measure was achieved when the activation function ReLU was used, with a dropout of 0.1 and the optimizer as rmsprop. Different values of epochs didn't make much difference unless they were dropped below the value of 2000. The addition of hidden layers didn't make much difference in performance in terms of accuracy for this particular problem but it took longer to train the model with additional layers. Overall, the accuracy of neural network was way better than accuracy of linear regression and logistic regression for all cases.