Project 2: Machine learning techniques to find similarity between handwritten samples

Arinjoy Bhattacharya
Department of Physics
University at Buffalo, SUNY
Buffalo, NY 14214
Person # 50168806
arinjoyb@buffalo.edu

Abstract

Handwriting recognition has been used by criminal detection agencies intensively. Such a dataset has been provided to test the accuracy of working of linear regression as well as classification methods like logistic regression and Neural networks.

1 Introduction

Linear regression is a very powerful tool which has been very effective in deep learning and information retrieval. But, all datasets may not be equally efficient in handling the provided model. Through this report, a comparison has been made between regression and classification methods and their parameters trained.

2 Model

2.1 Model: Dataset

The used dataset uses “AND" images samples extracted from CEDAR Letter dataset. Image snippets of the word “AND" were extracted from each of the manuscript using transcript-mapping function of CEDAR-FOX. The features of the images were extracted two ways:

- Human Observed Dataset: The features for the word were extracted by a human document examiner into two settings:
  - Feature Concatenation: containing 18 features.
  - Feature Subtraction: containing 9 features

  The entire dataset consists of 791 same writer pairs and 293,032 different writer pairs(rows).
- GSC Dataset: Gradient Structural Concavity algorithm generates 512 sized feature vector for an input handwritten “AND” image:
  - Feature Concatenation: containing 1024 features.
  - Feature Subtraction: containing 512 features

  The entire dataset consists of 71,531 same writer pairs and 762,557 different writer pairs(rows).

2.2 Model: Partitioning the dataset

Like in any other Machine Learning problem, we implement the method of training, validation and testing. So we partition the dataset into three parts:

- 90% training
- 5% validation
- 5% testing

These three datasets are distinct and don't overlap with each other.

2.3 Model: Linear regression model

In this case the Gaussian radial distribution model to map the linear regression model.

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

Here, ϕ(x) are the Gaussian radial basis functions. Then a probabilistic approach is implemented to find the error associated with the output. Now the output is a normal distribution and the Error function can be given as:

$$E = \sum_{i=1}^{N} (t_i - w^T \phi(x_i))^2$$

2.3.1 Model: Error function definition

The Error function in the matrix form can be represented as:

$$E = \frac{1}{2}\left(t^T - w^T \phi^T(x)\right)(t - \phi(x)w)$$

where t and ϕ(x) represent follows:

$$t = \begin{bmatrix} t_1 \\ \vdots \\ t_n \end{bmatrix} \text{ and } \phi(x) = \begin{bmatrix} \phi(x_{11}) & \cdots & \phi(x_{1n}) \\ \vdots & \ddots & \vdots \\ \phi(x_{n1}) & \cdots & \phi(x_{nn}) \end{bmatrix}$$

Each row in the ϕ(x) matrix correspond to a set of relu basis functions corresponding to t_1.

2.3.2 Model: Regularization

There is need for regularization so as to prevent overfitting. In general, there is a regularization term added to the weighted error term so that the weights term minimizes the error along with the regularizer term.

$$E(w) = E_D(w) + \frac{1}{2}\lambda w^T w$$

Our closed form solution with a regularizer has a $\lambda\mathbf{I}$ term added to it where $\mathbf{I}$ is the identity matrix, and λ is our regularizer term.

2.3.3 Model: Clustering

The dataset is too huge to work with it by itself. Thus it is broken up into smaller matrices using k-means clustering so that they can be fitted into Gaussian basis functions. Then these basis functions are used for regression by utilizing their mean values and the Gaussian spreads. The means are denoted by μ and the spread as a matrix called Σ.

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\sigma_1^2} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \boldsymbol{\sigma_D^2} \end{pmatrix}$$

2.3.4 Model: Stochastic Gradient Descent solutions

This algorithm decreases the weights based on a particular learning rate which is provided by the user. Therefore, the change in the learning rate affects the change in the error function accordingly. The change in error function is inversely proportional to the change in the value

of learning rate.

$$\Delta w = -\eta \nabla E$$

2.4 Model: Logistic regression model

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

2.4.1 Model: Sigmoid function definition

In this case of classification, it uses a sigmoid function to predict the probabilities.

$$S(z) = \frac{1}{1 + e^{-z}}$$

2.4.2 Model: Cost function definition

To generate the probability function, the weights from each cases are taken and feed it into the sigmoid function to generate the probability class. But as can be observed, the Mean square error function can't be used here since the probability function is nonlinear. Hence, a better cost function needs to be implemented.

The cost function used here is given by:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$Cost(h_\theta(x), y) = -\log(h_\theta(x)) \qquad if\ y = 1$$

$$Cost(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad if\ y = 0$$

The cost function determines how the weights are updated for the model.

2.5 Model: Neural Networks

Neural networks are trained iteratively using optimization techniques like gradient descent. After each cycle of training, an error metric is calculated based on the difference between prediction and target. The derivatives of this error metric are calculated and propagated back through the network using a technique called backpropagation. Each neuron's coefficients (weights) are then adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold.

2.5.1 Model: Neurons

The inputs to each node are combined using a weighted linear combination. The result is then modified by a nonlinear function before being output. Mathematically, it can be shown by the equation.

$$z_j = b_j + \sum_{i=1}^{n} w_{i,j} x_i$$

, where $w_{i,j}$ represents the weight vector, x_i represent the input vector from each neuron and z_j gives the values in the hidden layer.

2.5.2 Model: Layers

There are mostly speaking three layers in the network, the input layer, the hidden layer and the output layer. The provided code has two hidden layers, so essentially speaking there are

total of four layers. In the hidden layer, this is then modified using a nonlinear function such as a relu,

$$f(x) = x^+ = \max(0, x)$$

to give the input for the next layer. This tends to reduce the effect of extreme input values, thus making the network somewhat potent to arbitrary values. The parameters $b_1, b_2, b_3, w_{1,2}$ ... are "learned" by training the data. The values of the weights are often confined to prevent them from blowing up. The parameter that restricts the weights in the code is known as the 'learning rate', which varies from 0.1 to 1.

2.5.3 Model: Weights

The weights take arbitrary values to commence with, and these are then updated utilizing the observed data. Thereupon, there's a part of unpredictability within the predictions made by a neural network. Therefore, the network is typically trained many times considering totally different random beginning points, and also the results area unit averaged. The number of hidden layers, and the number of neurons in each hidden layer, must be specified in advance.

2.5.4 Model: Activation functions

Activation functions takes the input from the hidden layer and modify the values to tune the model. Activation functions give neural networks to model complex non-linear relationships. By modifying inputs with non-linear functions neural networks can model highly complex relationships between features. Popular activation functions include relu and leaky relu.

2.5.5 Model: Loss functions

After the hidden layer, a loss function is implemented to check how well the model performs for a particular set of parameters. In the provided code, it is a cross entropy function whose slope gives us how the model actually is performing. So, depending on the performance of the model, the parameters are modified by the loss function.

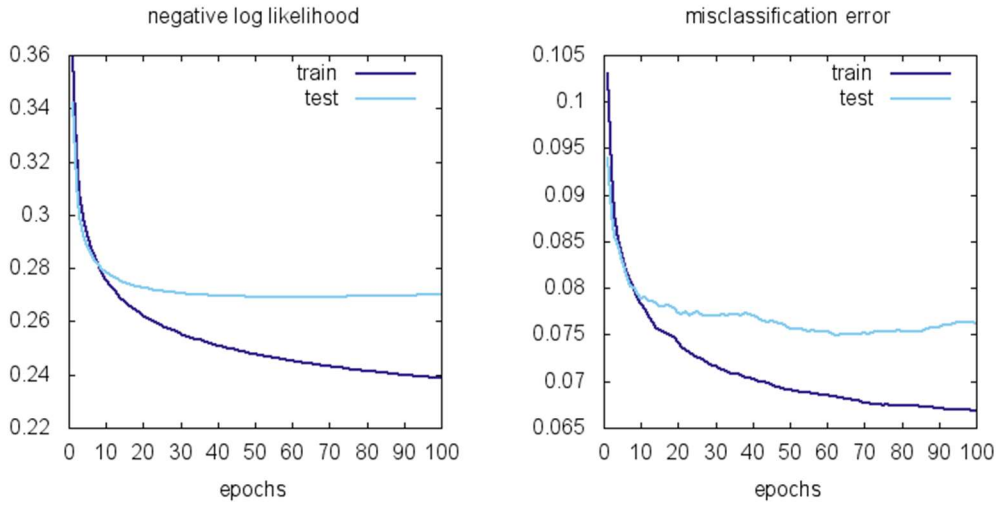

Figure 1: Range of possible loss values for cross entropy for a particular set of epochs.

2.5.6 Model: Optimization Methods

Optimization algorithms are used to back process the model and reduce the error in accuracy rate of the model. The method used in the provided code is gradient descent. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

$$\hat{C} = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2$$

2.5.7 Model: Forward and Backward Propagation

Essentially speaking, the two methods of propagation are complimentary to one another. The forward propagation involves calculating weights and multiplying them with the input, and finally using the activation function to pass it to the output layer. In case of backpropagation each weight in the network are adjusted in proportion to how much it contributes to overall error. The error function if differentiated with respect to particular hyperparameters and equated to zero to get our best probable accuracy.

3 Experiment

For the experiment, the linear regression model was trained with the various hyperparameters and are sequentially explained with plots below. Similarly, the logistic regression and the neural network was also trained as given below

3.1 Experiment: Linear Regression: different number of radial basis functions.

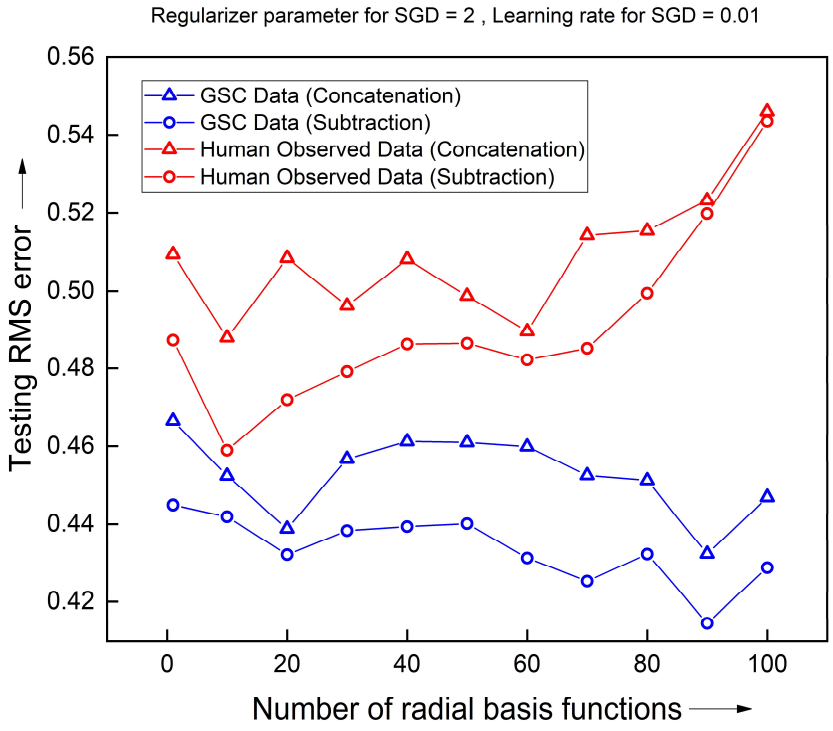

Figure 2: Testing E_{RMS} Vs number of radial basis functions which in turn is the number of clusters.

From the figure shown above, it clearly shows that increasing the number of basis functions leads to decrease of the root sum of square errors. Also, an important observation to be made here is that the subtraction features method has lesser E_{RMS} than the concatenation. A possible explanation of this can be that more number of data points lead to overtraining or overfitting the model. But on the other hand, GSC data has lesser error, which might be due to the fact that it is more distributed or in other words the training can be done better.

3.2 Experiment: Linear Regression: different values of learning rate of SGD.

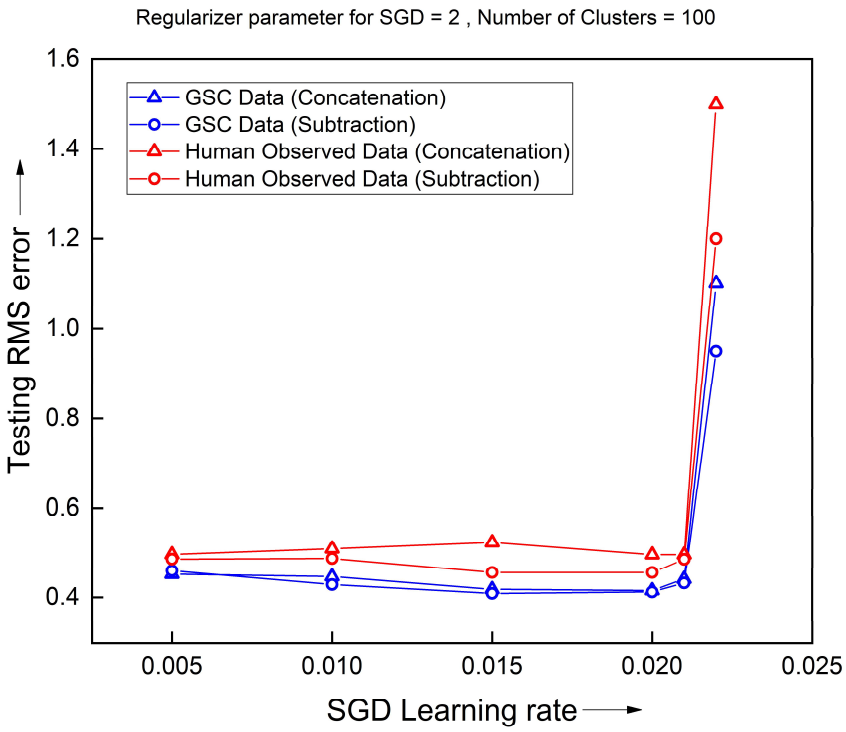

Figure 3: Testing E_{RMS} Vs different values of learning rate (η)

The learning rate is also detrimental in changing the error of the Error function which can be seen to blow up after the value of 0.02. Now learning rate basically determines how much should the error function of SGD decrease, or in other words how much should the weights be changed so as to get optimal error.

$$\nabla w = -\eta \nabla E \Rightarrow w_{new} = w_{prev} + \Delta w$$

Maybe, by changing the SGD regularizer to a lower value we can push the blowup to a bigger value of learning rate (η).

3.3 Experiment: Logistic Regression: different values of learning rate of SGD.

In case of Logistic, we measure accuracy to check whether our model is working or not. From the data which we have, and the results which we obtained we can easily say a logistic regression model is better suited to fit our data than linear regression. Accuracy decreases with increase in learning rate. Actually, it is basically like a bell-shaped. Trying out various learning rates, we basically find out a local maximum value of learning rate where the accuracy is maximum. In our case, it's around a value of 0.01.

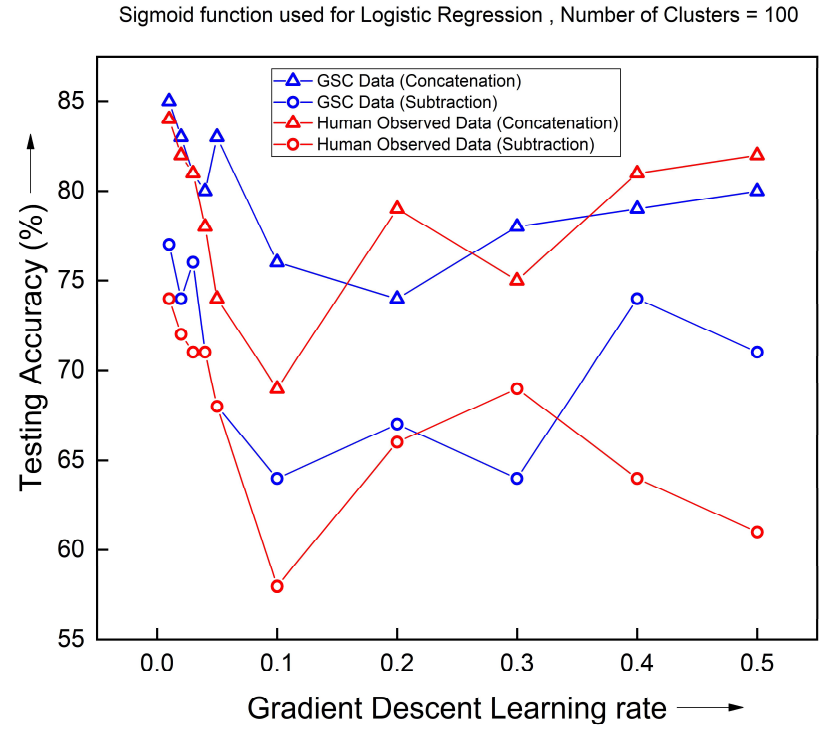

Figure 4: Testing Accuracy Vs different values of learning rate (η)

3.4 Experiment: Neural Networks: different values of learning rate.

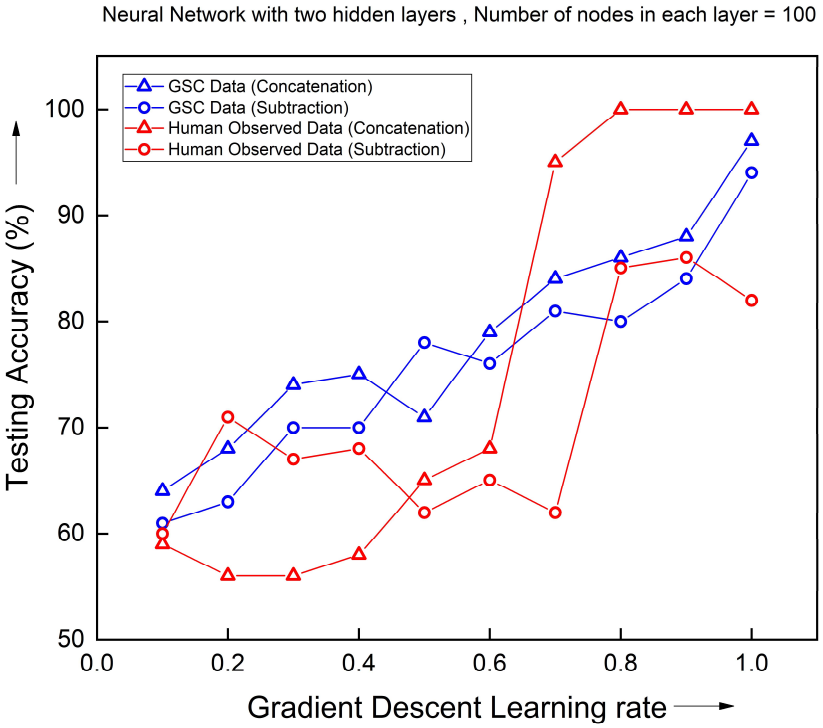

Figure 5: Testing Accuracy Vs different values of learning rate (η)

The learning rate also affects the accuracy in case of the classification implementation of the multilayer neural network. It steadily increases by increasing the learning rate.

3.5 Experiment: the perfect parameter

For the least RMS error, the following parameters seemed proper for Linear Regression:

- 100 or more relu basis functions (clusters in k-means)
- SGD learning rate of 0.01.
- SGD Regularizer of around 2.
- Number of iterations in the SGD calculation = 2000.

For the highest accuracy, the following parameters seemed proper for Logistic Regression:

- SGD learning rate of 0.01.
- Number of iterations in the SGD calculation = 2000.
- Sigmoid for updating the weights.

For the highest accuracy, the following parameters seemed proper for Neural Networks:

- 100 or more nodes in each layer.
- Learning rate of 1.
- Number of epochs = 5000.

4 Conclusion

Linear regression gives an average Testing accuracy of 50-55%, Logistic gives an average of 77-81% while two hidden layer neural network returns 99%. Thus we can safely assume that this is an example of a classification problem.

Acknowledgments

I thank the professor Dr. Srihari and the TAs for helping me complete this report.

References

[1] CM Bishop, TM Mitchell. (2014) *Pattern Recognition and Machine Learning*, Springer.