CSE574 Assignment 2 Report

Machine Learning Models: Handwriting Comparison

Tenzin Norden
UB person# 50096989
October 10, 2018

1 Introduction

In this project, three different machine learning models is used to solve handwriting comparison problem. The task is to find similarity between the known and the unknown handwritten "AND" samples using Linear Regression, Logistic Regression and Neural Network. The target value has a value '1' if the writers are same and '0' if the writers are different. This is a classification problem where the target value can have two values (0 or 1).

The given training data consists of a set of input features extracted from each hand written "AND" sample. The features are extracted using two different methods:
1. Human observed features: features are extracted by human document examiner experts manually
2. GSC features: Features are extracted using Gradient Structural Concavity (GSC) algorithm.

Model 1 is based on Linear Regression, where we map a set of input features **x** to a real-valued scalar target **y(x,w)**. Model 2 is based on Logistic Regression, where the problem is treated as binary classification problem, where the set of input features **x** is mapped to output using weights **y(x,w)** which is then classified into two classes (0 or 1) using a logistic function known as sigmoid. Model 3 is based on Neural Network to predict the input features **x** to outputs as binary value (0 or 1) and it is implemented using tensorflow framework.

2 Dataset

The dataset in this project uses "AND" images samples from CEDAR letter dataset. Two different datasets are used based on feature extraction process.

2.1 Human Observed Dataset

In Human observed dataset, the features for each image is extracted by a human document examiner. A total of 9 features are extracted from each sample images (HumanObserved-Features-Data.csv). The entire dataset consists of 791 same writer pairs (same_pairs.csv) and 293,032 (diffn-pairs.csv) different writer pairs (rows). The target is 1 for same writer pairs and 0 for different writer pairs.
The pairs are used to construct the required dataset.

In this project, datasets are constructed based on the same (791 pairs) and randomly picked 791 different writer pairs list. Image labels of the writers are compared in the same and different pairs list and their corresponding features are retrieved and processed two ways. (1) Features of the pairs are concatenated which results in 18 features. (2) Features are subtracted resulting in 9 features. The same and different pairs are then appended and randomly mixed with their target values as one whole dataset for features subtracted and concatenated datasets.

(1) For features concatenation, a dataset of 1582 sample pairs (rows) and 18 features (columns) are created.

(2) For features subtraction, a dataset of 1582 pairs (rows) and 9 features (columns) are created. Their corresponding target values are extracted as a separate matrix (1582,1).

All three machine learning models were performed on these two datasets.

2.2 GSC Dataset using Feature Engineering

Here for an input handwritten "AND" image, 512 features are generated using Gradient Structural Concavity algorithm. The features are stored in GSC-Features-Data.csv. The entire dataset consists of 71,531 same writer pairs and 762,557 different writer pairs. Similar to Human observed dataset, feature concatenation and feature subtraction datasets are constructed. In this project, only 5000 same writer and 5000 different writer pairs are used to decrease computing power and time consumption.
(1) Feature concatenation dataset have 1024 features (columns) and 10,000 sample pairs (rows).
(2) Feature subtraction dataset have 512 features (columns) and 10,000 sample pairs (rows).

2.3 Datasets

Four datasets are created after feature values and the Image Ids are extracted from the original csv data for same and different pairs. The datasets are processed into a Numpy matrix.

(a) Human Observed Dataset with feature concatenation
(b) Human Observed Dataset with feature subtraction
(c) GSC Dataset with feature concatenation
(d) GSC Dataset with feature subtraction

2.3 Data Partitioning

The four datasets and their corresponding target vectors are processed into Numpy matrix. The datasets are partitioned as 80% to a training set, 10% to a validation set and 10% to a testing set. The three datasets do not overlap. The training set is used to learn the weights or the parameters. The validation set is used to validate the learned weights and parameters. The test set is used at last to test the accuracy of the model with the learned weights.

3. Model

3.1 Model 1. Linear Regression

In this project, we model the linear regression function y($\mathbf{x}$,$\mathbf{w}$) using M basis functions and has the form:

$$y(\mathrm{x},\mathrm{w}) = \mathrm{w}^{\mathrm{T}}\phi(\mathrm{x}) = w_0\phi_0 + w_1\phi_1 + \cdots w_{M-1}\phi_{M-1}$$

where $\mathrm{w} = (w_0, w_1, \ldots., w_{M-1})$ is weight vector to be learnt from the training samples and $\phi = (\phi_0, \phi_1, .., \phi_{M-1})^{\mathrm{T}}$ is a vector of M basis functions. The M basis functions are non-linear functions of input variables and brings a non-linearity to the model. Therefore, M can represent the complexity of the model. We assume $\phi_0(\mathrm{x})$ = 1 for all inputs to introduce w_0 as the bias term in the model.

Here we chose the basis function as Gaussian radial basis function

$$\phi_j = \exp\big((x-\mu_j)^T \Sigma_j^{-1} (x-\mu_j)\big)$$

where μ_j is the center of the basis function and Σ_j decides how broadly the basis functions are spread. Each basis function turns the input vector into a scalar value.

1. K-means clustering is used to choose basis functions wherein the observations are clustered into M clusters, and their centroids are centers for Gaussian radial basis functions μ_j. The center of the Gaussian radial basis function μ_j is formed into a vector of MxN dimensions where **N** is number of features and **M** is a tunable hyperparameter number of basis function and it also determines the complexity of the model.

2. A uniform spread for all basis functions $\Sigma_j = \Sigma$ is considered in this project and it is constrained to a diagonal NxN dimension matrix where each diagonal element represents the variance σ_D^2 from each **N** feature columns.

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \cdots & . \\ \vdots & \sigma_2^2 \cdots & . \\ . & \cdots & \sigma_D^2 \end{bmatrix}$$

3. **Each basis function then turns the input vectors into a scalar value.** For example, the terms in the exponent of Gaussian radial basis function undergo matrix multiplications for every cluster 1 to M resulting into a scalar value as described below.

$$(x_i^1-\mu_1)^T \Sigma_j^{-1} (x_i^1-\mu_1) = \begin{bmatrix} (x_1^j-\mu_1) & (x_2^j-\mu_2) & \ldots\ldots & (x_i^j-\mu_j) \end{bmatrix} \begin{bmatrix} \sigma_1^2 & \cdots & . \\ \vdots & \sigma_2^2 \cdots & . \\ . & \cdots & \sigma_D^2 \end{bmatrix}^{-1} \begin{bmatrix} (x_1^j-\mu_0) \\ (x_2^j-\mu_1) \\ \vdots \\ (x_i^j-\mu_j) \end{bmatrix}$$

where for $\mathbf{x}_i^j$ is the value of feature **i** in **j**th training example. A matrix multiplication above is the form of **[1xN] [NxN] [Nx1]** resulting in a scalar output [1x1]. Each Gaussian radial basis function is a scalar value depending on the parameters such as its centers $\boldsymbol{\mu_j}$, variance $\boldsymbol{\sigma_D^2}$ and number of basis function **M** (model complexity).

4. After calculating the basis function $\phi(\mathbf{x}_n)$, it is then multiplied with the weights to perform linear regression. The function **y(x, w)** is still linear in **w** and $\mathbf{x}_n$ but non-linear with respect to $\phi(\mathbf{x}_n)$.

The output is assumed to have a probabilistic normal distribution with its mean as y(**x**,**w**) and noise β which is assumed constant. The likelihood function has the form:

$$p(\mathbf{t}|\mathbf{x}, \mathrm{w}, \beta) = \prod_{n=1}^{N} N(t_n|\mathbf{w}^T \phi(\mathbf{x}_n), \beta^{-1}) = \exp\left(-\frac{\sum_{i=1}^{N}[t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)]^2}{2\beta^2}\right)$$

where $\mathbf{x} = \{x_1, \ldots, x_n\}$ are the input samples and $\mathbf{t} = \{t_1, \ldots, t_n\}$ are target values. **Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error.**

$$E_D = \frac{1}{2}\sum_{i=1}^{N}[t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)]^2$$

5. In order to smoothen out the curve and prevent overfitting, a regularization term or a penalty term is added to the error function as follows.

$$E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_{\mathrm{w}}(\mathbf{w})$$

where λ is a tunable hyper parameter which governs the importance of the regularization term and the weight decay regularizer is

$$E_{\mathrm{w}}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathbf{T}}\mathbf{w}$$

The goal in this project is to minimize the error function while containing overfitting at the same time using regularization term.

6. **Stochastic Gradient Descent (SGD)**

In stochastic gradient descent approach the weights are randomly initialized $\mathbf{w}^{(\mathbf{o})}$ which is then updated after minimizing the first order differentiation of the error function ∇E. The updated weight is done as follows

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau)} - \eta^{(\tau)}\nabla E$$

where $\eta^{(\tau)}$ is the learning rate, which decides the step size of the gradient descent. The weights are updated over many iterations until the global minimum of the error function is reached. The gradient of the error function is given by

$$\nabla E_D = -(\mathbf{t}_n - \mathbf{w}^{(\tau)\mathrm{T}}\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$

And a regularization term is added to contain overfitting as follows.

$$\nabla E = \nabla E_D + \lambda \nabla E_W$$

where $\nabla E_W = \mathbf{w}^{(\tau)}$. $\boldsymbol{E_{RMS}}$ is again calculated after minimizing the error as

$$\boldsymbol{E_{RMS}} = \sqrt{2E(\mathbf{w}^*)/N_v}$$

We again tune the hyperparameters such as M, learning rate $\eta^{(\tau)}$, and λ and try to find the optimum value which give the lowest $\boldsymbol{E_{RMS}}$ value.

3.2 Model 2: Logistic Regression

Logistic Regression is very similar to linear regression where a set of input features **X** is mapped to a real valued scalar target y(x,w) using weights. Unlike linear regression, the output y(x,w) is treated under a logistic function called sigmoid which brings non-linearity to the model and maps the output into values between 0 and 1. The output is further classified into binary values (0 or 1) by setting proper thresholds at 0.5.

The output scalar is obtained as,

$$y(\mathrm{x}, \mathrm{w}) = \mathrm{w}^{\mathrm{T}} x = w_0 x_0 + w_1 x_1 + \cdots w_{M-1} x_{M-1}$$

where $\mathrm{w} = (w_0, w_1, \ldots., w_{M-1})$ is weight vector to be learnt from the training samples and $x = (x_0, x_1, .., x_{M-1})^{\mathrm{T}}$ is a vector of input features.

a. Logistic Function: Sigmoid

Sigmoid function is an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1 as shown in the Figure. The sigmoid function $\boldsymbol{\sigma}(\boldsymbol{z})$ has the form

$$\boldsymbol{\sigma}(\boldsymbol{z}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{z}}}$$

where **z** is $y(\mathrm{x}, \mathrm{w}) = \mathrm{w}^{\mathrm{T}} x$ in logistic regression.

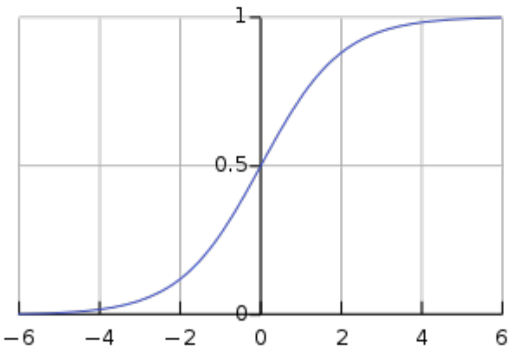

Figure 1. Sigmoid Function (source: https://en.wikipedia.org/wiki/Sigmoid_function)

b. Error Function:

The output scalar value $y(\mathrm{x}, \mathrm{w}) = \mathrm{w}^{\mathrm{T}} x$ is treated with sigmoid function as,

$$\boldsymbol{S}(\boldsymbol{y}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})}}$$

We can now define an error function based on the log likelihood to find the best values for the weights **w**. Error function is defined as

$$E(w) = \{-t \,.\, log(S) - (1 - t).\log(1 - S)\}/m$$

where "t" is the target vector and m are the number of input samples or length of target the vectors. The error function for different target values are

for t = 0, $E(w) = \{-\log(1 - S)\}/m$

for t = 1, $E(w) = \{-t \,.\, log(S)\}/m$

c. Gradient Descent:

To optimize the value of the weights, the error has to be minimized. It can be done by minimizing the derivative of the error function with respect to each weight.

$$\nabla E_D = \sum_{N=1}^{m} X_n^T . \{S(x,w)_n - t_n\}/m$$

Where With the error function and its gradient descent defined for logistic regression, the stochastic gradient descent algorithm is essentially same like that of Linear Regression algorithm as defined previously. The weights are randomly initialized $\mathbf{w}^{(o)}$ which is then updated after minimizing the first order differentiation of the error function ∇E. The updated weight is done as follows

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau)} - \eta^{(\tau)}\nabla E$$

By changing the hyperparameters such as learning rate, the error is minimized. The updated optimized weight matrix is then used to predict the output. $\mathbf{w}^{(\tau+1)}$ is multiplied with the input training features dataset $\mathbf{x}$. The trained output is again fed into sigmoid function, which outputs values from 0 to 1. A threshold of 0.5 is set to classify all the output values to either 0 or 1.
The optimized parameters are used on the validation dataset to validate. Finally, the testing dataset is used to test the model and find the accuracy of the model.

3.3 Model 2: Neural Network

In this machine learning model, two hidden layer Neural Network algorithm is implemented with tensorflow framework to solve this classification problem of handwriting matching.

The input features dataset is divided into 90% Training set and 10% Testing dataset. The target vector comprises an array of 1's and 0's.

(a) The model is set up in tensorflow which is three layers deep with two hidden layers and one output layer. Figure 1 shows a neural network model with 1 hidden layer. Tensorflow placeholder for input variable with width M, Number of features and output variable with width 1 is created. Number of neurons in the hidden layer is set to 100 and can be changed.

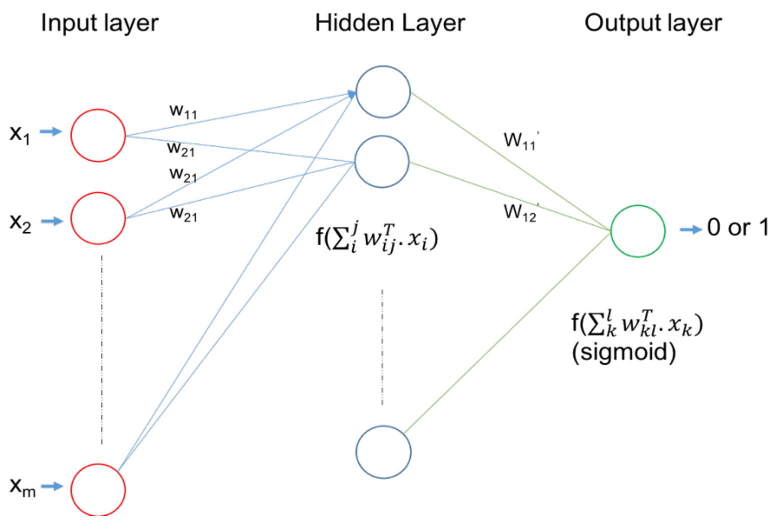

Figure 2. Two layers deep neural network with one hidden layer and 10 input neurons. Relu Activation function used at hidden layer and softmax activation is used at the output layer

(b) The weights for the input, hidden and output neurons are randomly initialized to a normal distribution. *In the hidden layer, an activation function is applied on the sum of the dot products of input vectors (x) and their corresponding weights (w^T), which gives the output of that layer.* An activation function is used to make neural network non-linear and limits the output signal to a finite value, for eg, between 0 and 1.

$$\mathrm{Relu}(f) = (\textstyle\sum_i^j w_{ij}^T . x_i , 0)$$

In this model, **relu (rectified linear unit)** activation function is used in the hidden layer which is just f(x) = max(0, x). Relu limits the output to be either 0 or x and it can only be used in hidden layers of the neural network.

(c) The output of the hidden layer after applying relu function is set as input for the output layer of our neural network model. Once again, the input to this layer is multiplied with a set of random weights and their sum is fed into **sigmoid** activation function which computes probabilities for different output classes in the limit of 0 to 1.

(d) The output of the output layer is then compared with the actual known output. The error is computed using cross entropy error with logits function as we have classification problem. Steps so far can be called forward propagation.

(e) **Backpropagation** is then performed using **GradientDescent** optimizer to minimize the error. It propagate backwards in the network to compute gradient of error function with respect to weights and then optimize weights using Gradient Descent optimization to reduce error. Gradient descent is then used to update and optimize weights until a global minimum of the error function is reached. Tensorflow framework does all of these automatically. **Our model trains via backpropagation.**

(f) **Learning rate** is a hyperparameter that determines the steps of a gradient descent. A very small learning will lead to slow convergence while large learning rate might overshoot the minima and can cause loss function to fluctuate and even diverge. In this project, we experiment the performance our model by changing the learning rate from 0.001 to 1.

(g) To train the model, we run the tensorflow model graph for 2000 epochs, where one **epoch** is equivalent to one forward and one backward propagation of all the training data. In this model, we use the **batch size**, the number of training data in one forward and backward propagation, as 128. So, the total number of iterations in the model is approximately 35,156 iterations. The output is then classified into four classes after all the iterations or training.

(h) After batch and epoch training, the testing is done for the testing data.

(i) The output label is then compared with the known target values of the testing data and the accuracy of the model is computed.

4 Experiment

4.1 Linear Regression

In Linear Regression Model, E_{RMS} is measured by tuning various hyperparameters for the four datasets: Human Observed and GSC datasets features concatenation and subtraction. The problem is a classification problem, so measuring the accuracy of the output using Linear regression is not the best approach. The accuracies measured are about 50% for all the datasets, which is why it is not recorded.

4.1a. E_{RMS} vs λ

In the first experiment, to obtain the lowest E_{RMS} value, I tuned the regularization coefficient 'λ' (Lambda) in Stochastic Gradient Descent solution of the linear regression model for the four different datasets. λ is tuned with the values 0.01, 0.05, 0.1, 0.5, 1, 5, and 10. Other hyperparameters such as number of basis function M and learning rate are kept constant to find the most effective regularization term for the four datasets.

1. For both Human Observed datasets with Feature concatenation and Feature subtraction, it is observed that E_{RMS} is relatively small at around 0.5. The behavior is similar for both human observed datasets. E_{RMS} increases slowly with increasing λ as shown in figure 2a.
2. For GSC dataset with Feature concatenation and Feature subtraction, the E_{RMS} increases with increasing λ value as shown in figure 2b. A smaller value of E_{RMS} of around 0.4 is achieved at smaller values of λ below 0.5. However, E_{RMS} increases more rapidly with increasing λ as compared the human observed datasets.

It appears that smaller values of $\lambda < 0.5$ contains overfitting better for the current set up of fixed parameters.

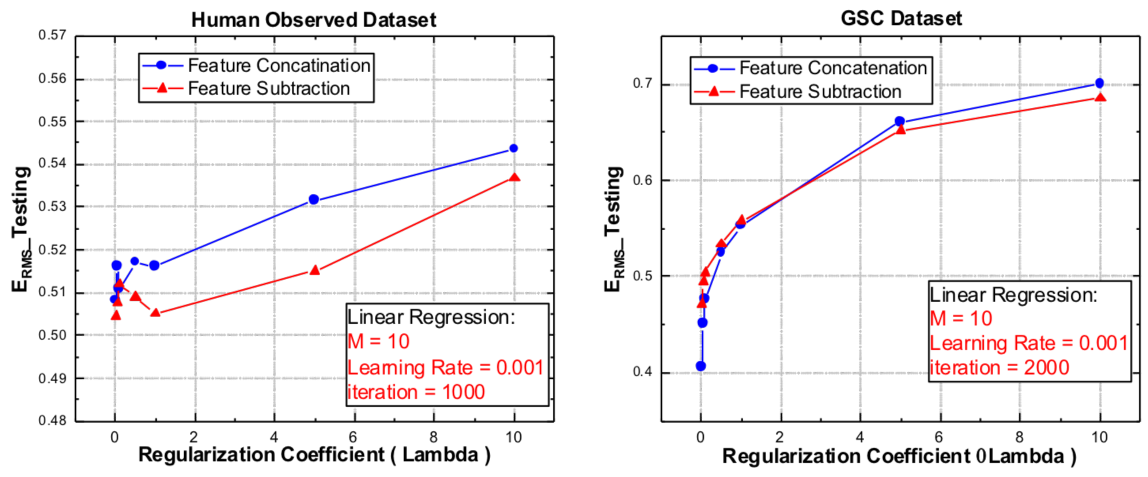

Figure 3. E_{RMS} vs λ. E_{RMS} from Stochastic Gradient Descent is scaled on the right vertical axis (red triangles).

4.1b. E_{RMS} vs M

In this experiment, E_{RMS} is measured as a function of M (number of clusters) or model complexity as shown in figure 4. In Human observed dataset, the E_{RMS} is recorded around 0.5 and it increases with increasing values of M for both feature concatenation and feature subtraction as shown in figure 4a. In figure 4b, E_{RMS} **vs M** for GSC dataset is plotted, where I observed that E_{RMS} does not change much with increasing M. The values for E_{RMS} for all the datasets is relatively small around 0.5.

It is clear that by increasing **M**, we are also automatically increasing the value of other hyperparameters such as number of clusters and variance. In doing so, we are maximizing the likelihood function, and hence, minimizing E_{RMS}.

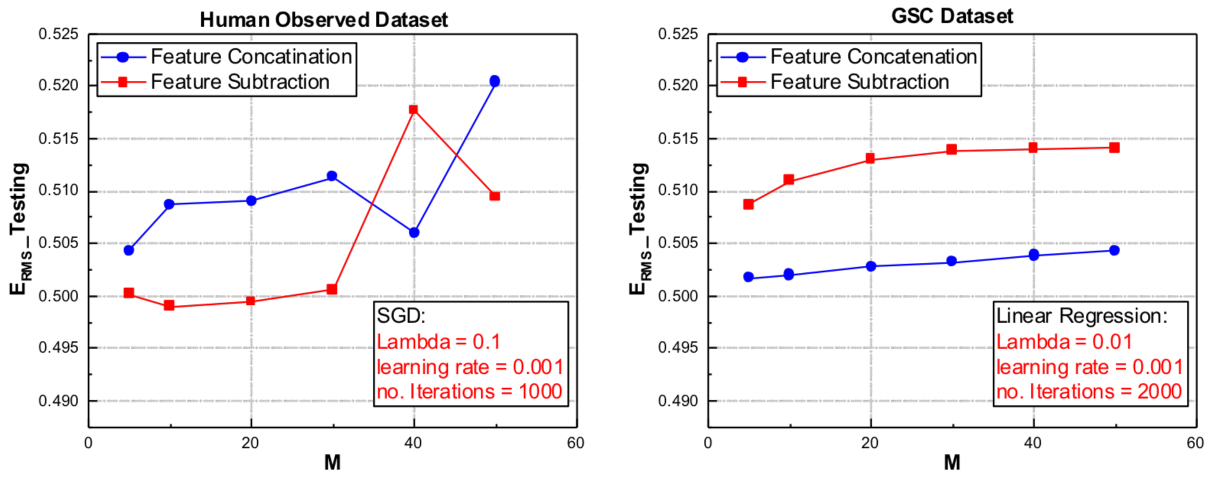

Figure 4. $\mathbf{E_{RMS}}$ for testing data plotted as function of M. (a) $\mathbf{E_{RMS}}$ vs M for Human observed dataset and (b) $\mathbf{E_{RMS}}$ **vs M** for GSC dataset.

4.1c E_{RMS} vs Learning Rate

Here E_{RMS} is plotted as a function of learning rate 'η' tuned from 0.001 to 0.05 as shown in figure 3. In Stochastic Gradient Descent Solution, learning rate decides the step size of the gradient descent which makes it a very important parameter. Finding proper learning rate is of utmost importance in in Gradient descent approaches because small learning rate may cost lot of computing power and time while large learning rate may over shoot the global minimum and the solution may not converge.

For human observed datasets, E_{RMS} is increasing with increasing learning for both features concatenation and feature subtraction as shown in figure 5a. For GSC dataset, E_{RMS} decreases with increasing learning rate unlike Human observed dataset as seen in figure 5b. The difference in the behavior in the two datasets could be due to the number of features.

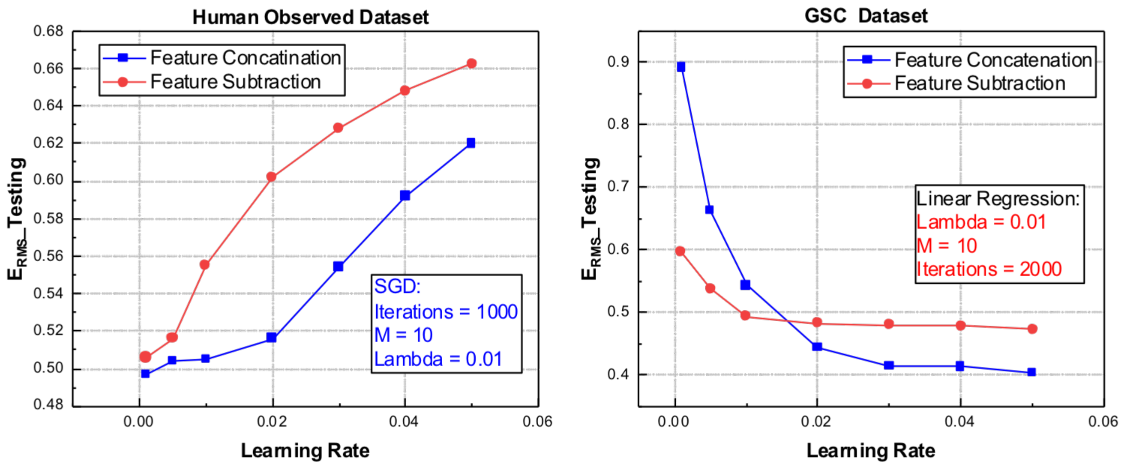

Figure 5. $\boldsymbol{E_{RMS}}$ vs Learning rate for (a) Human Observed Dataset and (b) GSC Dataset.

4.2 Logistic Regression

In this second model to solve the handwriting matching problem, logistic regression is used. Logistic regression is more suitable for this classification machine learning problem. Accuracies of the testing dataset is measured by tuning various hyperparameters such as learning rate and regularization coefficient λ for all the four datasets.

4.2a. Accuracy vs λ

For human Observed Dataset, feature concatenation shows relatively higher accuracies when compared to feature subtraction dataset as seen in figure 5a. The same trend can be seen in the GSC dataset, but with relatively higher accuracies as shown in figure 5b. The accuracies do not seem to change so much with different values of λ.

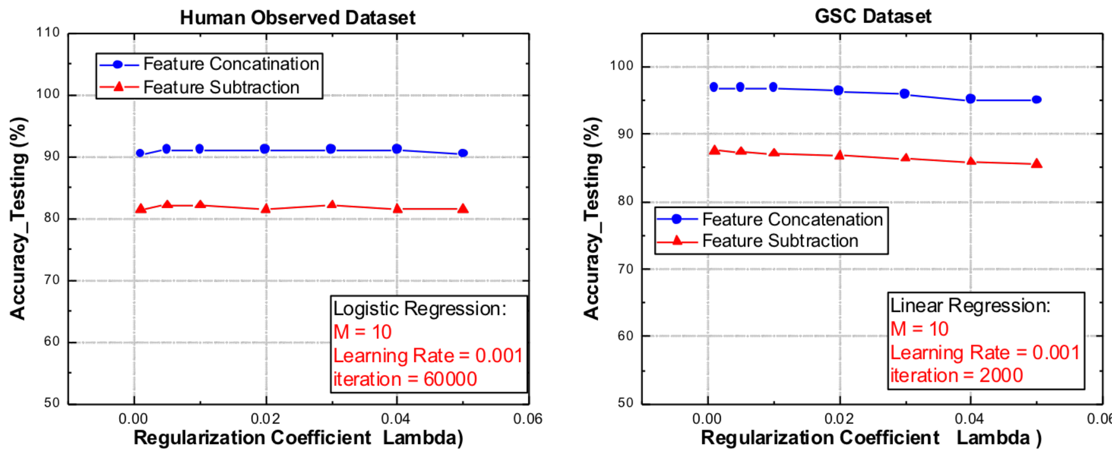

Figure 6. Accuracy is plotted as function of λ for feature concatenation and subtraction Human (a) Observed Dataset and (b)GSC dataset.

From the plot, it can be concluded that accuracy of the model is related to the number of features of the dataset. The feature concatenated datasets having more features than features subtraction datasets, clearly shows higher accuracies. The highest accuracy is seen in the GSC dataset with feature concatenated which has the greatest number of features of 1024. The lowest accuracy of is observed in feature subtraction of Human Observed dataset.

4.2b. Accuracy vs Learning Rate

In this experiment, Accuracy is recorded for different Learning Rate 'η' from 0.001 to 0.05 as shown in figure 7. For Human Observed Dataset, Accuracy decreases drastically for learning rates higher than 0.1. For learning rate below 0.1, the accuracy for human observed feature concatenation is about 90% and 80% for feature subtraction.

On the other hand, in GSC Dataset, feature concatenation really good with accuracy around 98% for all the used values of learning rate values used. However, the GSC feature subtraction dataset has accuracy about 85% for learning rate = 0.02 and below, but the performance deteriorates for higher learning rates.

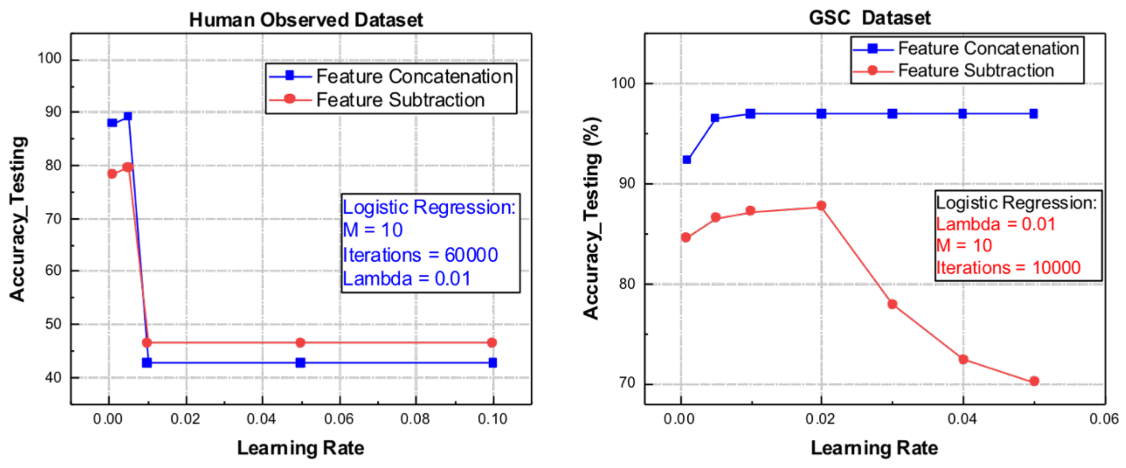

Figure 7. Accuracy vs Learning Rate. (a) Human observed Dataset with Feature Concatenation (blue) and feature subtraction (red). (b) GSC dataset with Feature concatenation (blue) and feature subtraction (red).

5. Neural Network

In third model, Neural Network machine learning model is implemented using TensorFlow framework with two hidden layers. Neural Network well suited for classification problems and high performance is expected. Number of neurons used in the two hidden layers is 100.

In figure 8, Accuracy is plotted as a function of learning rate for human observed and GSC datasets. In Human observed feature concatenation dataset, the accuracy of the model is really high at around 98% and even reached 100% for learning rate = 0.005. On the other hand, the accuracy is much lower for human observed feature subtraction as seen in figure 8a.

In GSC Dataset, the accuracy for both feature concatenation and feature subtraction is very high. GSC feature concatenation have accuracies of 99% for all learning rates used in the experiment. GSC feature subtraction dataset shows a slightly lower accuracy of around 95%.

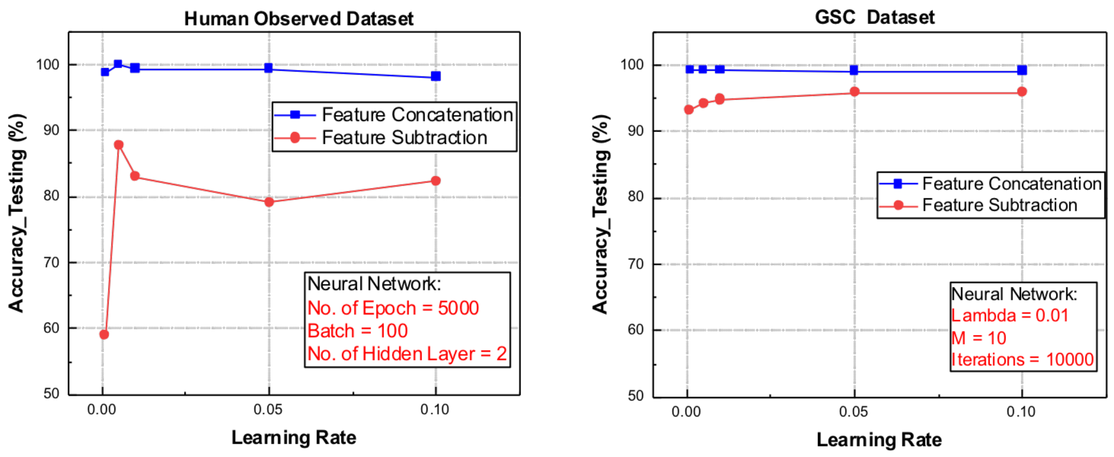

Figure 8. Accuracy vs Learning Rate for Neural Network model. (a) Human observed dataset, and (b) GSC dataset.

6. Conclusion

Handwriting recognition is a classification problem where pairs of handwritten 'AND' images are given a dataset. The features are extracted by Human experts and GSC algorithm. A set of four datasets are created which are Human observed features concatenation and subtraction, and GSC dataset with features concatenation and subtraction. Three model, linear regression, logistic regression and Neural Network is used to solve the problem of matching pairs of handwriting images using their extracted features. The target values can be either 1 for same pairs or 0 for different pairs. The output of the models were classified into 0 or 1 which makes it a classification problem.

$\boldsymbol{E_{RMS}}$ is measured form Linear regression model by changing various hyperparameters but it is not suitable for the problem in hand which is a classification problem. Accuracies are measured using Logistic regression and neural network models. It can be clearly observed that the GSC datasets with feature concatenated with greatest number of features performs best as compared to Human observed dataset with feature subtraction which has the least number of features.

In all models, random weights are chosen initially which then updated after every iteration of linear regression until optimal performance of the model is reached.