CSE574: Introduction to ML(Fall'18)

Project 2: Handwriting Recognition using various models

Vishal Shivaji Gawade
Department of Computer Science
University at Buffalo
Buffalo, NY,14226
vgawade@buffalo.edu

Abstract

The goal is to find similarity between the handwritten samples of the known and the questioned writer by using linear regression, logistic regression and neural network. Each instance in the CEDAR "AND" training data consists of set of input features for each hand- written "AND" sample. The features are obtained from two different sources:
1. Human Observed features: Features entered by human document examiners manually
2. GSC features: Features extracted using Gradient Structural Concavity (GSC) algorithm.
The target values are scalars that can take two values {1: same writer, 0: different writers}. Although the training target values are discrete we use linear regression to obtain real values which is more useful for finding similarity (avoids collision into only two possible values).

1.Preprocessing Data

First, we preprocessed the data of human observed features and GSC features. We have total 9 features for human observed image sample and 512 features for GSC image sample. To get the processed data sets we are using subtraction and concatenation technique on each two pairs we are comparing and creating processed data set for our models.

1. Human Observed Dataset with feature concatenation – 18 features -> HODfeaturesConcat.csv and 1 target-> HODtargetConcat.csv, total 19 columns
2. Human Observed Dataset with feature subtraction – 9 features -> HODfeaturesSub.csv and 1 target->HODtargetSub.csv, total 10 columns
3. GSC Dataset with feature concatenation – 512+512 features -> GSCfeaturesConcat.csv and 1 target->GSCtargetSub.csv, total 1025 columns
4. GSC Dataset with feature subtraction – 512 features -> GSCfeaturesConcat.csv and 1 target->GSCtargetSub.csv, total 513 columns

We are using numpy's shuffle function to shuffle the data. After this we are partitioning our data into training, validation and testing sets i.e. 80%,10% and 10%. Since we have a greater number of different pairs compare to same pairs, to balance the data we are taking same number of data points from each set so that we can train our model without any bias. Once we got our raw data, we are training our model using linear regression, logistic regression and neural network. We tune the hyper parameters for each model and select the best ones who give us minimum ERMS for linear regression and maximum accuracy for logistic regression and neural model. Finally, we test our machine learning scheme on the testing set i.e. on 10% .

Google drive link of processed data folder-https://drive.google.com/drive/folders/1s4W_Btwxx9K8-7KrwHdZbYvsOsemDFo1

2. Train using Linear Regression

We trained our model using solution which is given as,

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

Steps-

1. To find the Stochastic Gradient Descent solution for w, we need to find value of delta E.
2. After we get the value of delta E, we find delta w using which we get w for current data point.

3. We use this current w, called as w now to calculate next weight matrix for next data point as shown in above formula. For first data point we take a random initial value $w^{(0)}$ .In our code we used weight matrix $w^{(0)}$ as below,

```
W=np.random.normal(loc = 0.0, scale = 0.02, size = (M2,))
W_Now          = np.dot(256, W)
```

4. While calculating each weight matrix, we calculate root mean square value for each data point.
5. Finally, we take minimum root mean square value from all RMS values of data points in our code and say that this is the root mean square error for whole dataset.
6. We are training on len(TrainingTarget) for human observed data set and 800 for GSC dataset for calculating RMS because of computational issue.

3. Python code for linear regression

Python libraries used: from sklearn.cluster-KMeans, numpy , csv, math, matplotlib, pandas

3.1. Network Setting: The following hyperparameters combination gave me best accuracy and least RMS for linear regression solutions.

1.Linear regression for HOD subtraction-
Input: M = 9; Lambda = 2; learning Rate= 0.01
Output:

```
1266
----------Gradient Descent Solution--------------------

M =  9
Lambda  =  2
learningRate=  0.01
E_rms Training   = 0.50557
E_rms Validation = 0.50494
E_rms Testing    = 0.50483
Accuracy of testing data    = 54.14013

RawTarget = GetTargetVector('HODtargetSub.csv')
RawData   = GenerateRawData('HODfeaturesSub.csv')
```

Figure 1: Output (HOD Sub solution)

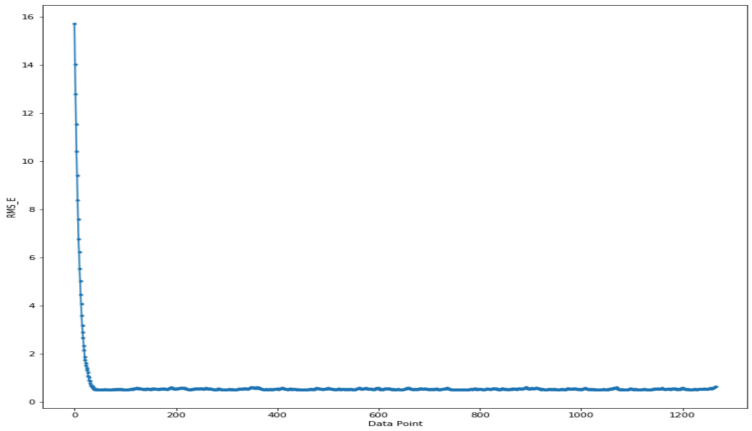

Figure 2: ERMS (HOD Sub solution)

2.Linear regression for HOD concatenation-
Input: M = 9; Lambda = 2; learning Rate= 0.01
Output:

```
1200
----------Gradient Descent Solution--------------------
linear-
M =  9
Lambda  =  2
learningRate=  0.01
E_rms Training   = 0.5011
E_rms Validation = 0.49698
E_rms Testing    = 0.50305
Accuracy of testing data     = 52.86624

RawTarget = GetTargetVector('HODtargetConcat.csv')
RawData   = GenerateRawData('HODfeaturesConcat.csv')
```

Figure 3: Output (HOD Concat solution)

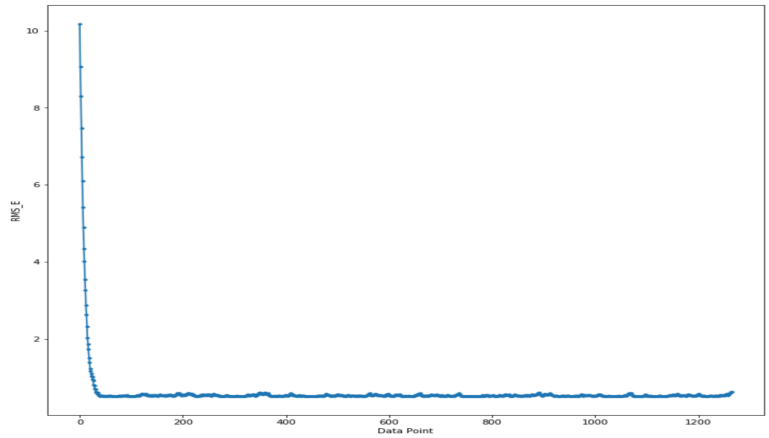

Figure 4: ERMS (HOD Concat solution)

3.Linear regression for GSC concatenation-
Input: M = 18; Lambda = 7; learning Rate= 0.01
Output:

```
114450
----------Gradient Descent Solution--------------------
linear-
M =  18
Lambda  =  7
learningRate=  0.01
E_rms Training   = 0.55589
E_rms Validation = 0.55737
E_rms Testing    = 0.55675
Accuracy of testing data     = 50.9612

RawTarget = GetTargetVector('GSCtargetConcat.csv')
RawData   = GenerateRawData('GSCfeaturesConcat.csv')
```

Figure 5: Output (GSC Concat solution)

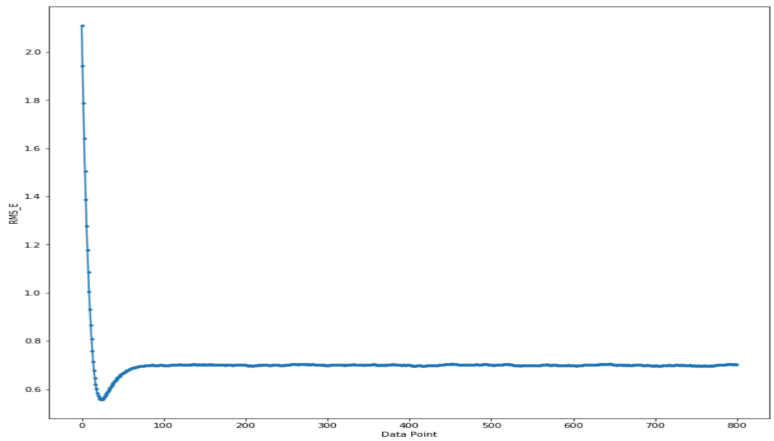

Figure 6: ERMS (GSC Concat solution)

4.Linear regression for GSC Subtraction-
Input: M = 9; Lambda = 8; learning Rate= 0.01
Output:

```
----------Gradient Descent Solution--------------------
linear-
M =  9
Lambda  =  8
learningRate=  0.01
E_rms Training   = 0.48674
E_rms Validation = 0.48611
E_rms Testing    = 0.48688
Accuracy of testing data    = 63.9986

RawTarget = GetTargetVector('GSCtargetSub.csv')
RawData   = GenerateRawData('GSCfeaturesSub.csv')
```

Figure 7: Output (GSC Sub solution)

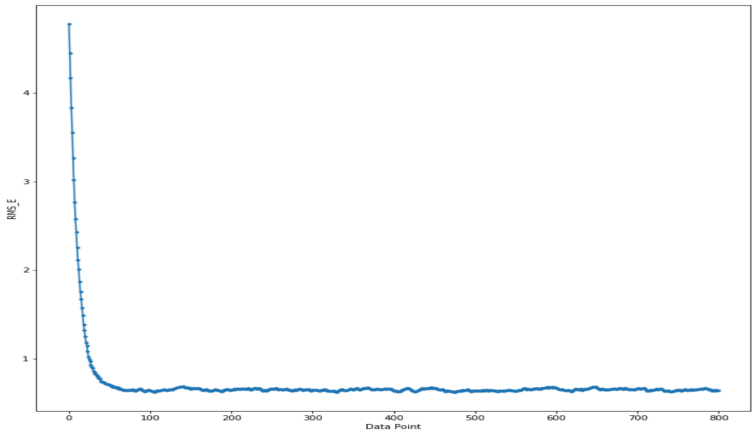

Figure 8: ERMS (GSC Sub solution)

Conclusions after trying different hyperparameters tuning-

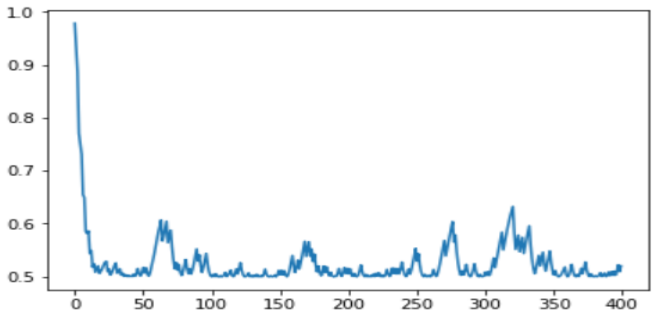

Figure: Hyper parameter tuning effect on ERMS

- If we use high value of M that means a greater number of clusters. If we use a greater number of clusters which will lower the variance, and if variance is lower that means Big Sigma value would be lower. All of this leads to increased value of basis function. Because of this root mean square value gets decreased as we increase value of M.
- If we increment value of learning rate we see that it directly affects the root mean square error. But as we go on incrementing value of learning rate, we see distortion in ERMS graph. So we should keep our learning rate value 0.01 for better performance from our model.
- If we increment value of regulariser we see that for SGD when lambda was 0.01 RMS value was nearly 10. But once we keep on incrementing lambda value it comes below 1. And after that we see very little effect on value of RMS. So we can say that when we keep regulariser value very low we can expect very high RMS error for our problem.

3.2. Code understanding -

Q. Why don't we use different Mu values for training, validation, testing data?
Ans-We don't change Mu because we are using it for training our model. So, if we change it for validation/testing it will give us wrong results. And we calculate Mu using entire data set. For this reason, we don't use different Mu values.

Q. Why we used 3 and 200 to generate big sigma value?
Ans-We can use any value to multiply big sigma. It just that we have to check how multiplying value affects Erms. We are using 3 and 200 to scale the variables.

Q. In SGD solution, can we initialise weights to zero?
Ans-We can initialise W_ now with any values. We can even pick it up randomly. In our current solution we used np.random.normal to generate weight matrix as initial W_now for SGD.

Q. Why use K-means clustering?
Ans-We use K-means clustering to generate M basis functions. There are other clustering algorithms but we used K-means because it assumes that clusters are multidimensional spheres and it can return cluster centroids. Other algorithms just assign a label to every data point is used. Using K-means we can find centroids which we will use to define our regression basis function.

Q. What is big sigma matrix?
Ans-Big sigma is matrix called variance matrix over each feature array for training datasets. We use its inverse to calculate Phi value, which is used to calculate basis function.

Q. Can we calculate big sigma for each cluster and use it in our solution?
Ans-Yes, we can find big sigma for each cluster. But if we find the covariance of entire dataset, the covariance will be more effective. Also, the chances of determinant being zero will be less as compare to calculating it for each cluster as we have filtered out those columns of features based on entire dataset. This is why we kept big sigma common for every dataset.

Q. How number of clusters we use affect the basis function?
Ans-If we go on increasing number of clusters, it will reduce the value of variance in clusters. Variance is our big sigma so big sigma will be lower. So, our basis function value would be higher.

Q. How can we tune Mu hyperparameter?
Ans- If we increase the number of basis functions, it needs more mu by clustering. So that way it gets changed. Another way is by shuffling (this is what we used for our model) and splitting the dataset without clustering for a fixed M.

Q. Why do we add constant to data value of GenerateRawData function?
Ans-Because to implement our model we need to find the inverse of matrix. But for GSC dataset lot of column values are zero, which leads to matrix being singular. To avoid this and to find inverse of matrix we added some noise to data.

4. Train using Logistic Regression

We trained our model using solution which is given as,

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

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

Our given problem is of binary logistic regression. We are going to use Sigmoid function to predict result.

5. Python code for logistic regression

Python libraries used: from sklearn.cluster-KMeans, numpy , csv, math, matplotlib, pandas

5.1. Network Setting: The following hyperparameters combination gave me best accuracy and least RMS for logistic regression solutions.

1. Logistic regression for HOD subtraction-
   Input: M = 2; Lambda = 1; learning Rate= 0.01
   Output:

```
----------Gradient Descent Solution--------------------

M =  2
Lambda  =  1
learningRate=  0.01
E_rms Training   = 0.49987
E_rms Validation = 0.49997
E_rms Testing    = 0.49891
Accuracy of testing data    = 54.14013
```

Figure 9: Output (HOD Sub solution)

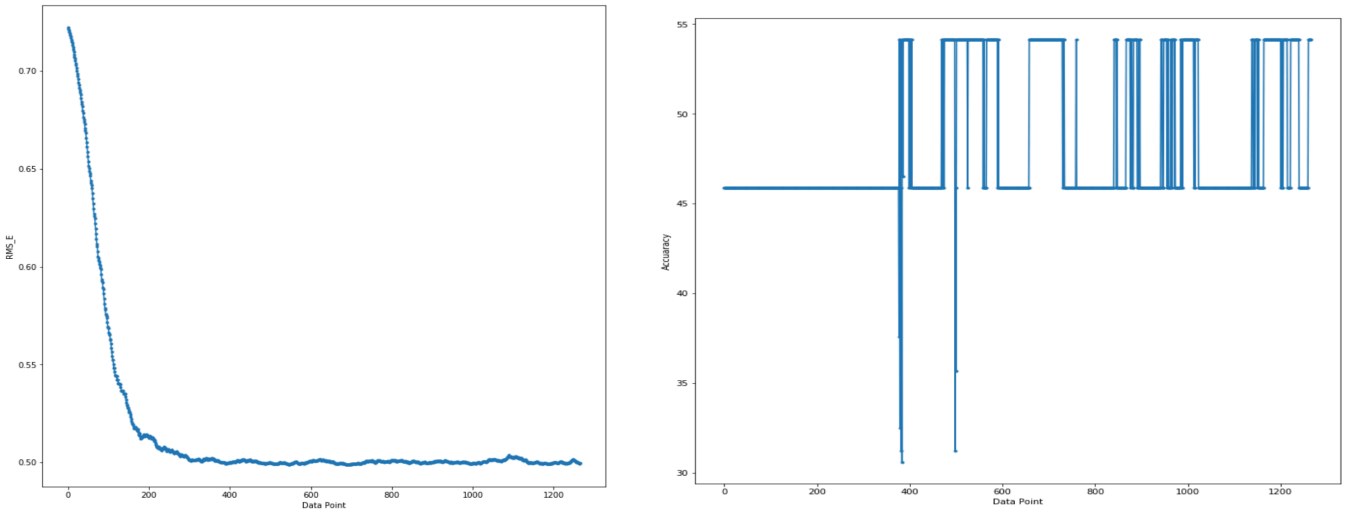

Figure 10: ERMS and Accuracy (HOD Sub solution)

2. Logistic regression for HOD concatenation-
   Input: M = 2; Lambda = 1; learning Rate= 0.01
   Output:

```
1266
----------Gradient Descent Solution--------------------

M =  2
Lambda  =  1
learningRate=  0.01
E_rms Training   = 0.49963
E_rms Validation = 0.49905
E_rms Testing    = 0.49999
Accuracy of testing data    = 50.95541
```

Figure 11: Output (HOD Concat solution)

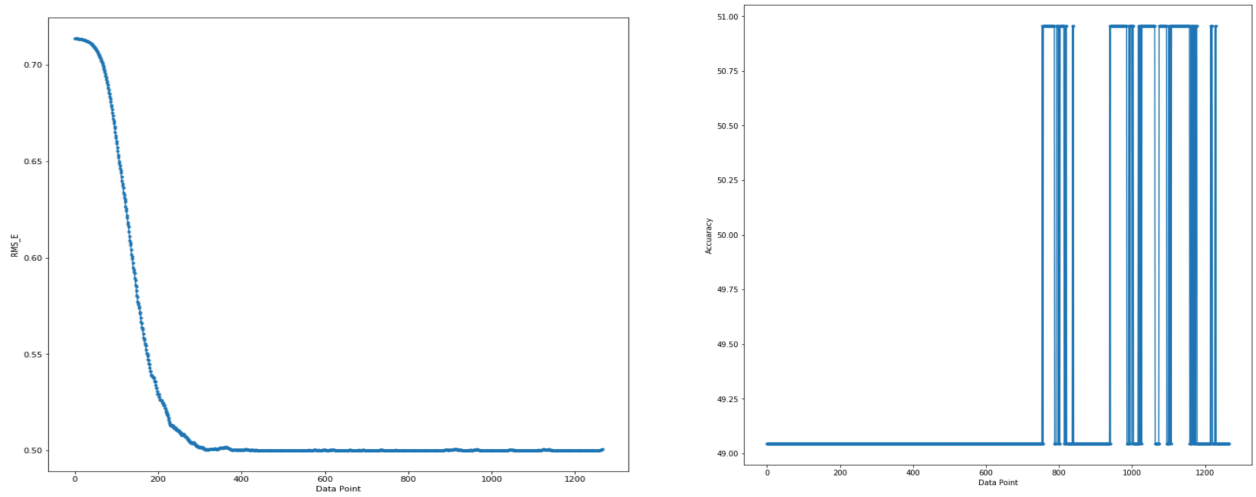

Figure 12: ERMS and accuracy (HOD Concat solution)

3. Logistic regression for GSC concatenation-
   Input: M = 7; Lambda = 2; learning Rate= 0.01
   Output:

```
----------Gradient Descent Solution--------------------

M =  7
Lambda  =  2
learningRate=  0.01
E_rms Training   = 0.55773
E_rms Validation = 0.55941
E_rms Testing    = 0.55838
Accuracy of testing data    = 51.16393
```

Figure 13: Output (GSC Concat solution)

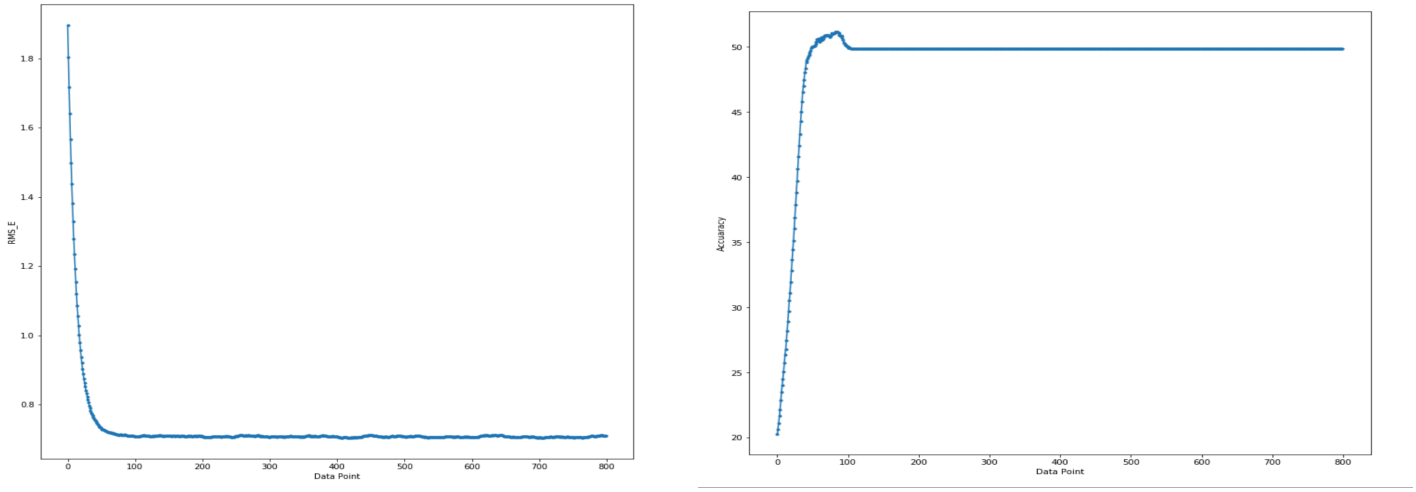

Figure 14: ERMS and Accuracy (GSC Concat solution)

4.Logistic regression for GSC Subtraction-
   Input: M = 9; Lambda = 8; learning Rate= 0.01
   Output:

```
----------Gradient Descent Solution--------------------

M =  9
Lambda  =  8
learningRate=  0.01
E_rms Training   = 0.48422
E_rms Validation = 0.48388
E_rms Testing    = 0.48442
Accuracy of testing data    = 64.36211
```

Figure 15: Output (GSC Sub solution)

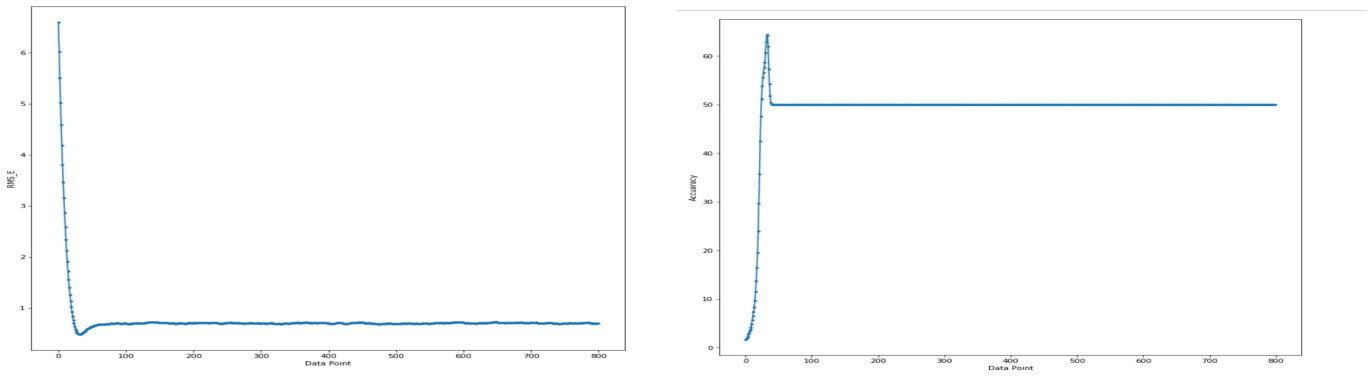

Figure 16: ERMS and Accuracy (GSC Sub solution)

Conclusions after trying different hyperparameters tuning-

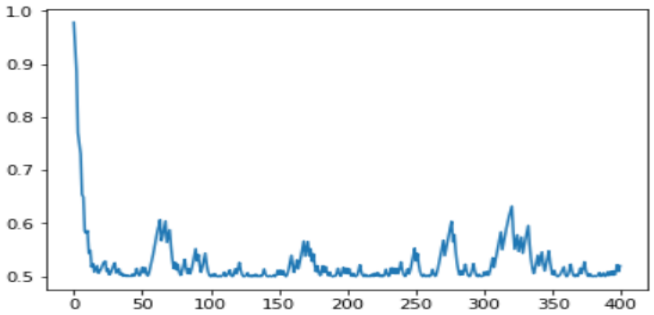

Figure: Hyper parameter tuning effect on ERMS

- Similar conclusions can be drawn for the logistic regression ERMS value like in linear regression.
- For learning rate value should be as less as possible.
- So we should keep value of learning rate 0.01

6. Train using Neural Network

Given a set of inputs X, we want to assign them to one of two possible categories same or different writer (0 or 1). Logistic regression models the probability that each input belongs to a particular category.

Neural Network consists of the following components

- An input layer, x
- An arbitrary amount of hidden layers-used two (first_dense_layer_nodes = 128, second_dense_layer_nodes = 64)
- An output layer, $\hat{y}$
- A set of weights and biases between each layer, *W and b*
- A choice of activation function for each hidden layer, σ. In this project, we'll use a **Sigmoid** after second layer and **Relu** after first layer as a activation functions.
- Used **Adadelta** Optimiser to optimise model.

7. Python code for Neural network

Python libraries used: from sklearn, numpy , csv, math, matplotlib, pandas, keras

7.1. Network Setting: The following hyperparameters combination gave me best accuracy for Neural network.
Common input for all datasets-

- drop_out = 0.3

- first_dense_layer_nodes = 128
- second_dense_layer_nodes = 64
- final_dense_layer_nodes = 1
- validation_data_split = 0.2
- model_batch_size = 512
- tb_batch_size = 128
- early_patience = 100

1. Neural network for HOD subtraction-
   Input: Input Size = 9; num_epochs=400
   Output:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_29 (Dense)             (None, 128)               1280
_________________________________________________________________
activation_29 (Activation)   (None, 128)               0
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 129
_________________________________________________________________
activation_30 (Activation)   (None, 1)                 0
=================================================================
Total params: 1,409
Trainable params: 1,409
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 400/400
1265/1265 [==============================] - 0s 9us/step - loss: 0.2009 - acc: 0.9296 - val_loss: 0.2123 - val_
acc: 0.9274
1582/1582 [==============================] - 0s 5us/step
Accuracy of model: 94.06%
```

Figure 17: Output (HOD Sub solution)

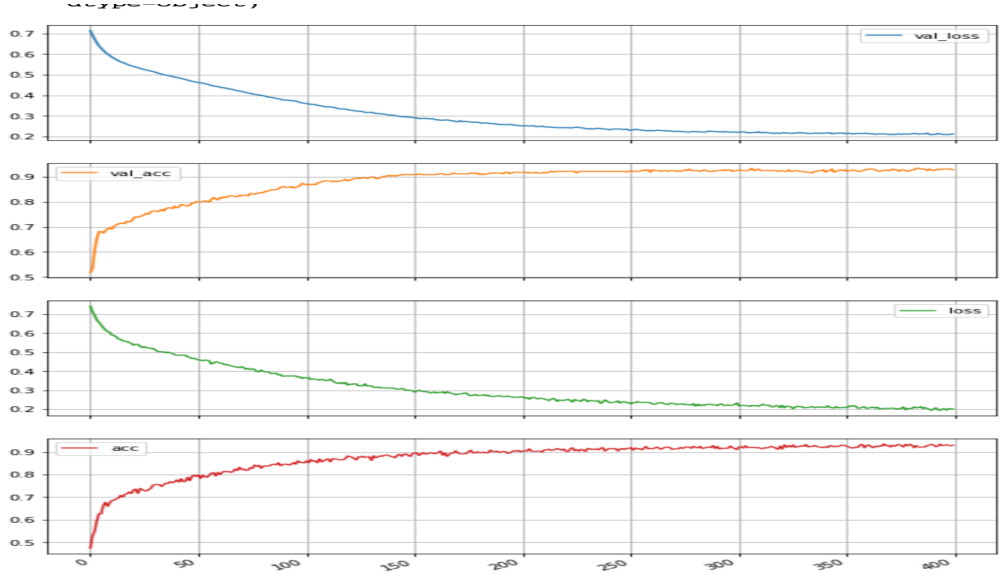

Figure 18: Diff Graphs (HOD Sub solution)

2. Neural network for HOD concatenation-
   Input: Input Size = 18; num_epochs=100
   Output:

Layer (type)	Output Shape	Param #
dense_17 (Dense)	(None, 128)	2432
activation_17 (Activation)	(None, 128)	0
dropout_9 (Dropout)	(None, 128)	0
dense_18 (Dense)	(None, 1)	129
activation_18 (Activation)	(None, 1)	0

Total params: 2,561
Trainable params: 2,561
Non-trainable params: 0

```
acc: 0.9748
Epoch 100/100
1265/1265 [==============================] - 0s 8us/step - loss: 0.1114 - acc: 0.9668 - val_loss: 0.1003 - val_
acc: 0.9621
1582/1582 [==============================] - 0s 3us/step
Accuracy of model: 96.71%
```

Figure 19: Output (HOD Concat solution)

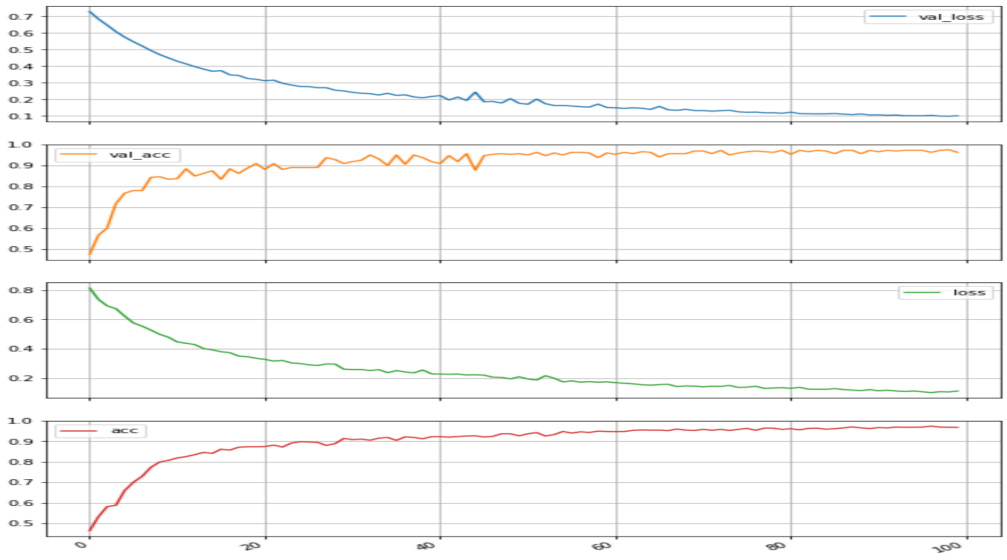

Figure 20: Diff Graphs (HOD Concat solution)

3. Neural network for GSC concatenation-
   Input: Input Size = 1024; num_epochs=100
   Output:

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	131200
activation_1 (Activation)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 1)	129
activation_2 (Activation)	(None, 1)	0

Total params: 131,329
Trainable params: 131,329
Non-trainable params: 0

```
val_acc: 0.9919
Epoch 100/100
114449/114449 [==============================] - 2s 16us/step - loss: 0.0013 - acc: 0.9997 - val_loss: 0.0462 -
val_acc: 0.9922
143062/143062 [==============================] - 1s 7us/step
Accuracy: 99.84%
```

Figure 21: Output (GSC Concat solution)

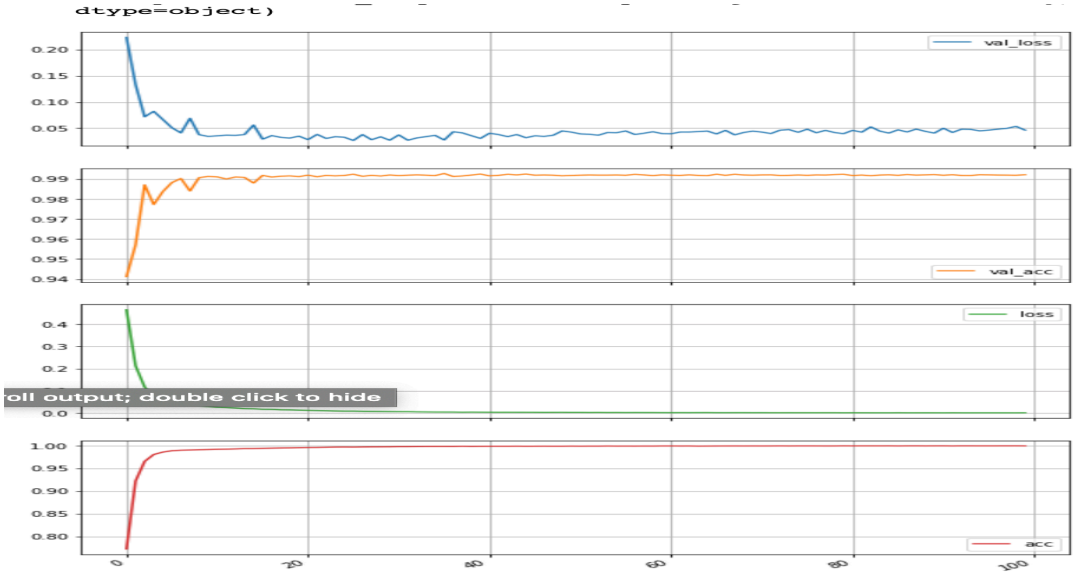

Figure 22: Diff Graphs (GSC Concat solution)

4.Neural network for GSC Subtraction-
Input: Input Size = 512; num_epochs=100
Output:

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 128)	65664
activation_5 (Activation)	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 1)	129
activation_6 (Activation)	(None, 1)	0

Total params: 65,793
Trainable params: 65,793
Non-trainable params: 0

```
114449/114449 [==============================] - 1s 9us/step - loss: 0.0037 - acc: 0.9988 - val_loss: 0.0369 -
val_acc: 0.9927
Epoch 100/100
114449/114449 [==============================] - 1s 9us/step - loss: 0.0037 - acc: 0.9988 - val_loss: 0.0373 -
val_acc: 0.9926
143062/143062 [==============================] - 1s 5us/step
Accuracy: 99.85%
```

Figure 23: Output (GSC Sub solution)

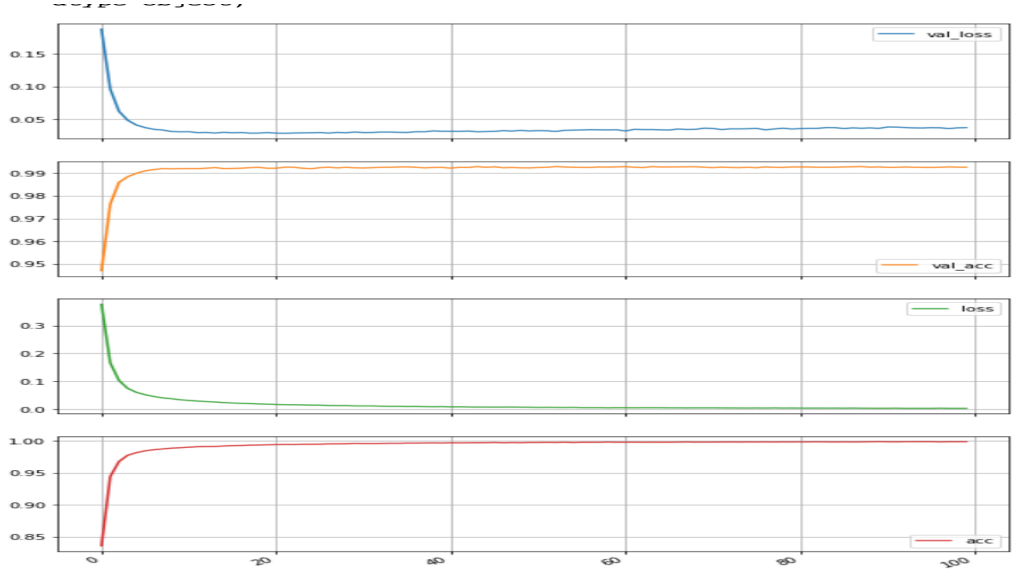

Figure 24: Diff Graphs (GSC Sub solution)

8.Difference between linear, logistic regression

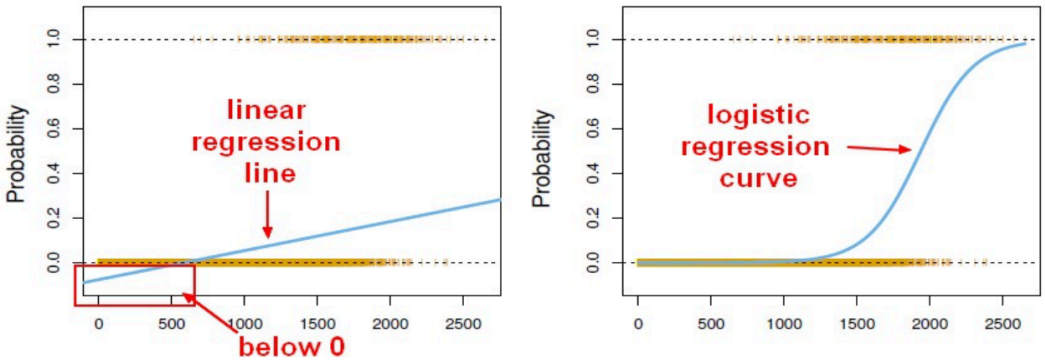

Figure 25: Linear vs Logistic Regression

Table 1: Difference between linear vs logistic regression

Linear regression	**Logistic Regression**
In linear regression, the outcome (dependent variable) is continuous. It can have any one of an infinite number of possible values.	In logistic regression, the outcome (dependent variable) has only a limited number of possible values.
Linear regression is used when your response variable is continuous. For instance, weight, height, number of hours, etc.	Logistic regression is used when the response variable is categorical in nature. For instance, yes/no, true/false, red/green/blue, 1st/2nd/3rd/4th, etc.
Linear regression gives an equation which is of the form Y = mX + C, means equation with degree 1.	Logistic regression gives an equation which is of the form Y = e^X + e^-X
In linear regression, the coefficient interpretation of independent variables are quite straightforward (i.e. holding all other variables constant, with a unit increase in this variable, the dependent variable is expected to increase/decrease by xxx).	In logistic regression, depends on the family (binomial, Poisson, etc.) and link (log, logit, inverse-log, etc.) you use, the interpretation is different.
Linear regression uses ordinary least squares method to minimise the errors and arrive at a best possible fit	logistic regression uses maximum likelihood method to arrive at the solution.
Linear regression is usually solved by minimizing the least squares error of the model to the data, therefore large errors are penalized quadratically.	Logistic regression is just the opposite. Using the logistic loss function causes large errors to be penalized to an asymptotically constant.

9.Conclusion

1. In simple linear regression a single independent variable is used to predict the value of a dependent variable. In multiple linear regression two or more independent variables are used to predict the value of a dependent variable. The difference between the two is the number of independent variables. Thus, when we have a problem in hand which have this kind of relation we use linear regression like we used in page ranking project 1.2
2. Logistic regression is used when the response variable is categorical in nature. For instance, yes/no, true/false, red/green/blue, 1st/2nd/3rd/4th, etc.

References

[1] Project 1.1 report
[2] Project 1.2 report
[3] https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac
[4] https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6
[5] https://gogul09.github.io/software/first-neural-network-keras
[6] https://stackoverflow.com/questions/12146914/what-is-the-difference-between-linear-regression-and-logistic-regression