Machine Learning CSE574

David Jegan Abishek Dominique
Person no 50290785
davidjeg@buffalo.edu

Abstract

The report provides a detailed insight on how Machine learning can be used to identify the similarity between two handwritten samples

1 Introduction

1.1 Handwriting similarity detection

This report addresses how we could identify the similarity between two handwritten samples using Linear regression, Logistic regression and Tensorflow methodologies. If the algorithm detects a match, then 1 is returned and similarily 0 is returned if there is dissimilarity between the two images. Though the results are scalar and discrete (0,1), we consider linear/logistic regression and tensorflow to obtain continuous value rather than a discrete on, to find the percent of match.

1.2 Approach

There are four stages in this problem

1. Create training data
2. Linear regression
3. Logistic regression
4. Neural Networks using Tensorflow

2 Methodology

2.1 Data Preparation

For the algorithm to learn we provide a training dataset. This data is crucial for the prediction as the model considers this data as Ground truth and tries to learn. The dataset here uses the image snippets of AND extracted from CEDAR Letter dataset. There are two dataset types, namely Human observed and Stochastic Gradient Concavity.

2.1.1 Human Observed dataset

The Human Observed dataset contains the details extracted from the image snippet by a human. Thus there are 9 features for a image, and for a pair of images there are 18 samples. There are 791 pairs of same writer and 293,032 pairs of different writer extracted from the image database.

2.1.2 Gradient Structural Concavity dataset

The Gradient Structural Concavity dataset contains the details extracted from the image snippet by a program. Thus there are 512 features for a image, and for a pair of images there are 1024 samples. There are 71,531 pairs of same writer and 762,557 pairs of different writer extracted from the image database.

2.1.3 Feature Data

As observed, the number of different writer pairs dominate, so in order to avoid model biasing we consider the same number of same/different writer records in both the samples, 1582 in case of Human observed dataset and 10000 in case of SGC dataset. The feature dataset concatenated with the target is shuffled to facilitate random training and test sample sets, so that the model is more robust.

2.1.4 Feature Concatenation

For regression, the more the data features, the more is the chances a system can learn about the model. So thus we add the 'n' features for the image pairs and pass it as '2n' features. Thus human and SGC has 18 and 1024 features respectively.

2.1.5 Feature Subtraction

To minimize the time and space complexity, we subtract the two 'n' features for the image pairs and pass it as 'n' features. Thus human and SGC has 9 and 512 features respectively.

2.1.6 Target Data

The actual result serves to test our model behaviour. Thus we divide the target values into percents of actual data called training, validation and testing result set, in such a way that the sets do not overlap. These values are then compared with the obtained results to compare the accuracy of our model.

Thus there are 4 different datasets in our paper.
1. Human Observed dataset with feature concatenation
2. Human Observed dataset with feature subtraction
3. SGC dataset with feature concatenation
4. SGC dataset with feature subtraction

2.2 Linear Regression using Stochastic Gradient Descent

In this approach, we build a linear regression model, which traverses the dataset incrementally, updating the values of weight and phi (basis function), calculating the minimal error function after each epoch to predict the target. Thus our algorithm moves towards lower error, updating weights accordingly to find the global cost minimum. Thus, the target can be determined using the equation,

$$t = w^T \Phi(x)$$

where $\Phi(x)$ is the basis function design matrix constructed using x (input set), w is the weight and t is the output (target).

2.2.1 Weight

Both the input and the target is provided in the training phase. So to optimize the model, we need to improve the accuracy of the weights, and weights are dependant on the error function. Thus in-order to change the weights, we reduce the error value between target and prediction for each input x. Therefore, the weight in next recursion is the sum of the weight in current recursion and an additive ∇ value.

Weights can be initialized randomly, so we consider the weights obtained from the previous stage. Considering W_{old} as weight obtained in a step i, we have the formula of W_{new} for step i+1 as,

$$W_{new} = W_{old} + \nabla W$$

$$\nabla W = -\eta \nabla E$$

$$W_{new} = W_{old} - \eta \nabla E$$

Here, η is the learning rate, and negative (-) sign denotes that the graph descends towards minima.

So we multiply the negative learning rate with ∇E and sum it with current weight (W_{old}) to obtain the new weight (W_{old})

2.2.2 Error function

Gradient descent iteratively minimizes the error function. The value of gradient on Error function gives us the direction of downhill. So we must differentiate our Error function to get desired solution.

The error function is defined as,

$$E = 0.5(t_i - w^T \phi(x_i))^2 + 0.5\lambda w w^T$$

where, t_i is the trainingtarget value for i, w is the current weight, $\phi(x)$ is the basis function design matrix for input x and λ is the regularization parameter.

We know that, differentiating E with respect to W, gives 0.

$$\nabla E = -(t_i - w^T \phi(x_i))\phi(x_i) + \lambda w$$

We calculate the weights based on the training dataset, and run our model against the validation and the test cases, to check the accuracy of our model.

This term is multiplied with the negate of learning rate, in each recursion.

2.2.3 E_{RMS}

This methods calculates the accuracy of our prediction with the actual target by identifying how different our model has deviated from the actual. If Erms equals 0 for train set, then it is overfitting. Erms is the square root of the sum of the difference square, divided by total predictions

2.2.4 Learning Rate

The steepness of the downhill is decided by the learning rate. We consider $\eta = \frac{1}{i}$ because we want our model to descends fast initially, and gradually slow down its desent.

2.2.5 Regularization

Inorder to avoid overfitting, we consider a regularization parameter (La), This is done while finding the weights. We have assumed the La variable randomly. The parameter reduces overfitting by adding bias to our estimate.

Also, regularizer is responsible for reducing the weight values and maintaining an optimal weight. This change is consistent with respect to the weights.

2.2.6 Accuracy

Based on the number of matches of Actual and Predicted values, we find the accuracy of the model.

2.2.7 Mini batch Gradient Descent

The usual SGD takes around 400 samples to update the weight iteratively for each sample. We ignore the rest of the sample, as the accuracy would not increase significantly. But SGC has around 10000 data samples and inorder to overcome this selective sample choosage, we use batch gradiant descent to use all the 10000 data samples. Whereas for the Human observed dataset we use the usual SGD method.
Since the delta error would be carried forward, we mitigate it by considering the mean of Training set divided by the batch size.

2.2.8 Gaussian Radial Basis Function

We extracted the initial weights using Radial Basis Function, then find the Sigma and variance for the Phi matrix. Basis function helps to define non-linear relationship between the target and input.

$$\phi_j(x) = e^{(-0.5(x-\mu_j)^T \Sigma^{-1}(x-\mu_j))}$$

On the same lines, we build Σ (covariance) matrix with the help of BigSigma functionality. μ is formulated by leveraging k-means clustering algorithm and x is extracted from the input 41 feature (R^{41}) dataset. The final size of ϕ_j is M*N, where M is the number of basis functions and N is the number of samples.

The ϕ_j matrix is built for train, validation and test data sets.

2.3 Logistic Regression

In this approach, we build a logistic regression model, which can predict the classification of the output by finding the optimal weights. The weights are found by iteratively updating the weights using the sigmoid and gradient values.

Thus, the genesis equation is formulated as,

$$\hat{y} = \sigma(w^T * x)$$

where x is the input set , w is the weight, ŷ is the output (target) and σ denotes the formula, $\sigma = 1/1 + e^{-x}$

2.3.1 Sigmoid function

Sigmoid is a mathematical function having a "s" shaped curve. This is significant because as the datasize increases, it tends to 0 or 1. Thus this acts like a normalization (filter) which gives singular and continuous value between [0,1] denoted by $\sigma = 1/1 + e^{-x}$.

2.3.2 Loss function

The loss function is calculated using the following formula,

$$L = -\sum_{i=1}^{N} y.loga + (1-y).log(1-a)$$

Where a denotes $\sigma(w^T * x)$, and the backward travel from L to w_i is done by using partial differential of L with respect to w_i . Thus $\Delta w_i = x_i(a-y)$, where a is the predicted target and y is the actual target.

2.3.3 Bias

We add a bias term to the feature set as an added noise. This is added to the feature matrix and the size is denoted by the number of sample records.

2.4 NN using Tensorflow

This model contains of leveraging Tensorflow to deploy neural network based prediction model with hidden layers. A network of entities named Neurons, are initialized and are responsible for running mathematical equations. There are layers of these neurons, and are connected to the next level via weights.

$$Neuron = \sum (Weight * Input) + Bias$$

2.4.1 Activation

The ML model consists of a network of neurons. Each neuron is linked with the next level via weights. The output for a neuron is called activation. Based on this output we decide an algorithm which is sensitive to

input and has faster computation time. Sigmoid is used here as an activation function, and it is used to normalize(filter) the data between [0,1].

2.4.2 Layers of NN

There are three types of layers, ie., Input, Hidden and Output layers. All three layers, input, hidden and Output are single layered in this example. The input and output layer directly map to the inputs and outputs respectively, whereas the hidden layer leverages the sigmoid method for the input layer and converts it into a context to be used by the output layer.

2.4.3 Weights

The weights connecting the neurons help to find the output. These values are updated via a method called Back-propagation. It can be initialized randomly or using a trained weight.

2.4.4 Learning Rate

The rate at which an model forgets its existing belief system is called learning rate. If the rate is high then the model adapts quickly and if its slow, it learns slowly.

2.4.5 Cost Function

This method helps to increase the accuracy of the model. The value must be low in order to mitigate errors. This is done by randomly finding the lowest minima for a function until we find a convergence point (Stochastic Gradient Descent).We use softmax_cross_entropy_with_logits cost function in this model.

2.4.6 Epoch and Blocksize

The number of iterations done on the training set is called epoch. Batchsize is the number of sample taken for processing by the model at a time.

3 Findings

3.1 Linear regression

3.1.1 Overall Accuracy

```
----------Linear Regression Gradient Descent Solution--------------------
human_concatenation Accuracy Training   = 49.447077409162716
human_concatenation Accuracy Validation = 53.164556962025316
human_concatenation Accuracy Testing    = 51.59235668789809
------------------------------------------------------------------------
human_subtraction Accuracy Training   = 50.39494470774092
human_subtraction Accuracy Validation = 52.53164556962025
human_subtraction Accuracy Testing    = 43.94904458598726
------------------------------------------------------------------------
gradient_concatenation Accuracy Training   = 54.675
gradient_concatenation Accuracy Validation = 56.65665665665666
gradient_concatenation Accuracy Testing    = 55.45545545545546
------------------------------------------------------------------------
gradient_subtraction Accuracy Training   = 73.075
gradient_subtraction Accuracy Validation = 73.07307307307308
gradient_subtraction Accuracy Testing    = 72.37237237237237
------------------------------------------------------------------------
```

Figure 1: Overall Predication accuracy (right) for Linear regression

We observe that with more number of features, the the error rate decreases and the accuracy increases proportionally. The reason for the accuracy in the 50 percentile mark is because of the datasamples, which is chosen at random. If they contain unique writers and equal same/different ratio, then the accuracy would increase accordingly.

There was no overfitting observed, as almost the accuracy of training, test and validation are on the same range.

3.2 Erms and Accuracy

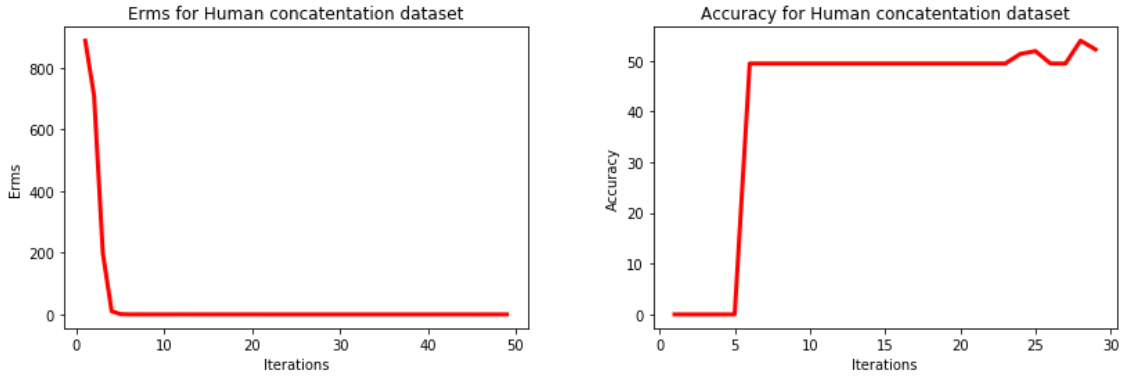

Figure 2: Erms (left) and accuracy (right) for Linear regression Human observed concatenated dataset

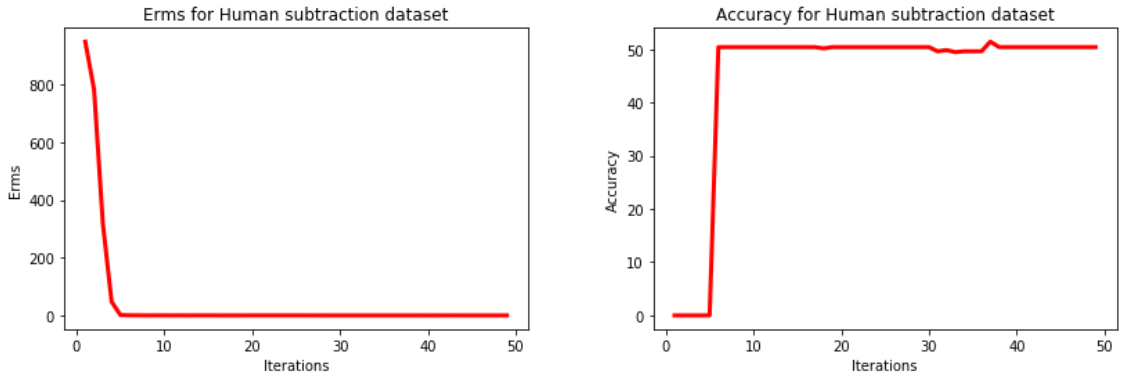

Figure 3: Erms (left) and accuracy (right) for Linear regression Human observed subtracted dataset

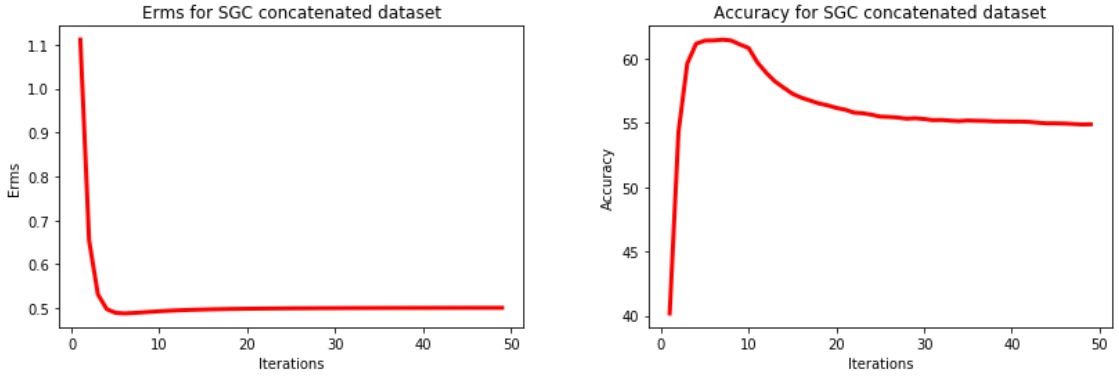

Figure 4: Erms (left) and accuracy (right) for Linear regression SGC concatenated dataset

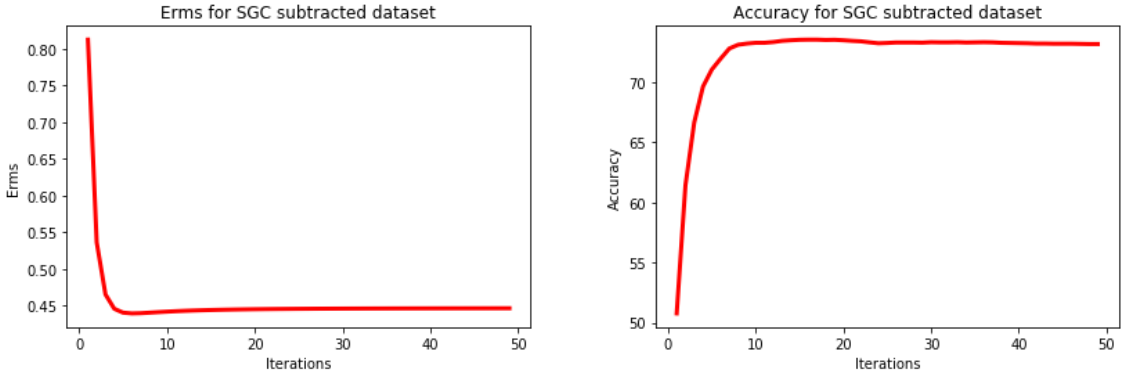

Figure 5: Erms (left) and accuracy (right) for Linear regression SGC subtracted dataset

From the graph, we find that for all 4 datasets, the Erms has decreased and has remained constant after some iterations. Similarily the prediction accuracy has also increased.

3.3 Logistic regression

3.3.1 Overall Accuracy

```
Logistic regression accuracy of Human concatenation is 0.5159235668789809
Logistic regression accuracy of Human subtraction is 0.4840764331210191
Logistic regression accuracy of GSC concatenation is 0.7497497497497497
Logistic regression accuracy of GSC subtraction is 0.5945945945945946
```

Figure 6: Overall Predication accuracy (right) for Logistic regression

We observe that with more number of features, the the error rate decreases and the accuracy increases proportionally. The reason for the accuracy in the 50 percentile mark is because of the datasamples, which is chosen at random. If they contain unique writers and equal same/different ratio, then the accuracy would increase accordingly. The accuracy is almost in the range of linear. There was no overfitting observed, as almost the accuracy of training, test and validation are on the same range.

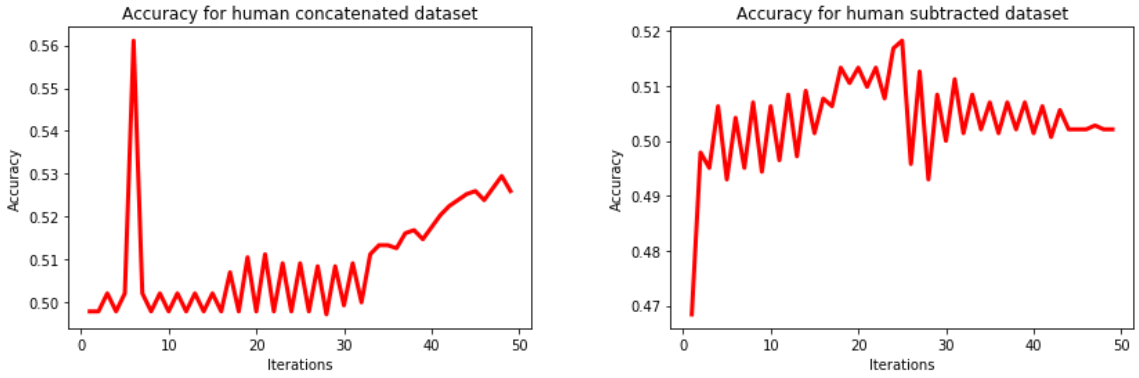

Figure 7: accuracy for logistic regression human concatenated (left) and subtracted(right) dataset

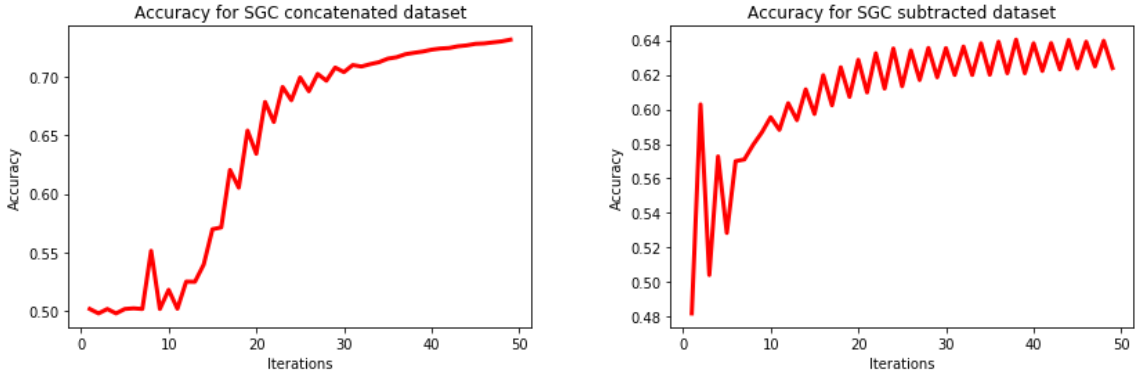

Figure 8: accuracy for logistic regression SGC concatenated (left) and subtracted(right) dataset

From the graph, we find that for all 4 datasets, the prediction accuracy has increased.

3.4 NN using Tensorflow

3.4.1 Overall Accuracy

```
Tensorflow Accuracy for Human Concatination  0.5164557
Tensorflow Accuracy for Human Subtraction  0.48607594
Tensorflow Accuracy for GSC Concatenation  0.7728
Tensorflow Accuracy for GSC Subtraction  0.6184
```

Figure 9: Overall Predication accuracy (right) for Neural networks using tensorflow

We observe that with more number of features, the the error rate decreases and the accuracy increases proportionally. The reason for the accuracy in the 50 percentile mark is because of the datasamples, which is chosen at random. If they contain unique writers and equal same/different ratio, then the accuracy would increase accordingly.

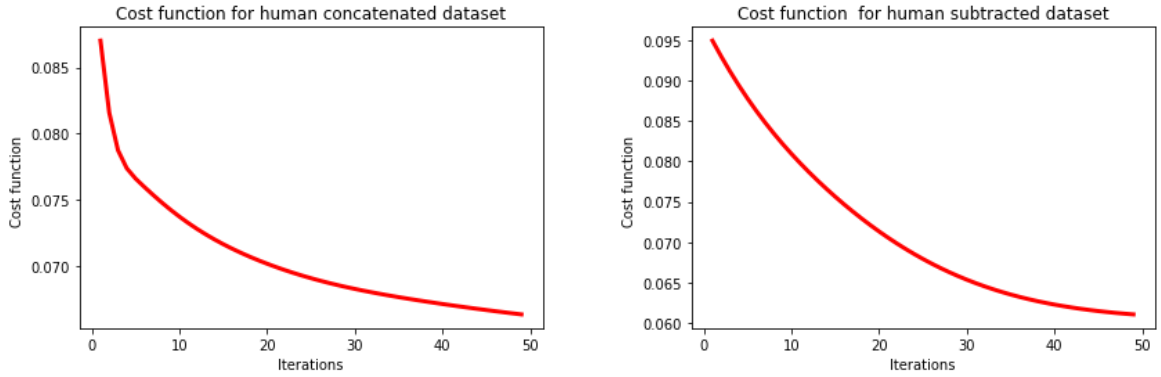

Figure 10: accuracy for logistic regression human concatenated (left) and subtracted(right) dataset

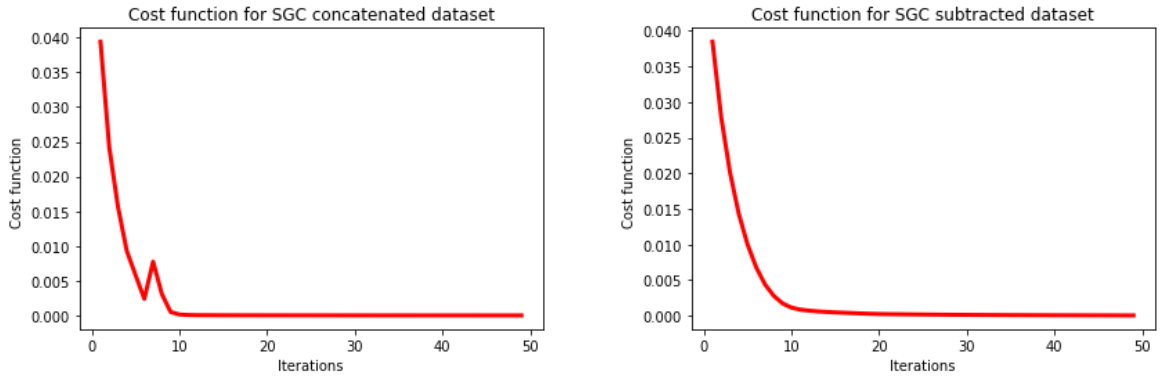

Figure 11: accuracy for logistic regression SGC concatenated (left) and subtracted(right) dataset

From the graph, we find that for all 4 datasets, the cost has decreased, thereby increases the accuracy.

4 Conclusion

From the models, the folowing inferences were observed

- We observe that SGC has better accuracy in all models compared the Human observed data. This is because there are many features for the system to correlate.
- Also, we can see that Concatenation gives better results than feature subtraction, because of the above mentioned increase in features for correlation.
- We see that for SGC dataset, the neural network implemented using tensorflow has better accuracy than logistic regression. likewise, Logistic had better accuracy over linear.
- Also we observe that for human observed dataset the accuracy for linear, logisitic and tensorflow is almost on the same range. ie., there is no significant improvement in accuracy. This is because of the reduced number of features.

References

1. https://cedar.buffalo.edu/ srihari/CSE574/Chap3/3.2-GradientDescent.pdf
2. https://cedar.buffalo.edu/ srihari/CSE574/Chap3/3.1-Regression-BasisFns.pdf
3. https://cedar.buffalo.edu/ srihari/CSE574/Chap4/4.3.2-LogisticReg.pdf
4. https://beckernick.github.io/logistic-regression-from-scratch/
5. https://github.com/soerendip/Tensorflow-binary-classification