Handwriting comparison using different machine learning models

Shruti Bendale
Person Number: 50289048,
School of Engineering and Applied Sciences,
University at Buffalo, Buffalo, NY, 14214
shrutitu@buffalo.edu

Abstract

The problem to be solved in this project is to distinguish two samples of handwriting from each other, with help of machine learning techniques. The main technique used is the comparison between two samples and classifying if they are written by the same person (match) or not (no-match). The binary classification problem is then tackled with a few alternatives to better understand it. First by linear regression, then by the machine learning techniques such as logistic regression and a neural network.

The raw data was initially preprocessed to create datasets that can be used for performing regression. Multiple linear regression using gradient descent and logistic regression were performed on the data and finally, the results were evaluated and compared. The evaluation method used for the three models was the Accuracy and Root Mean Square (RMS) error. The results of the alternative algorithms gave a better understanding of the problem.

1. Introduction

Handwriting comparison is a common task in forensics document analysis. Given two samples of the same word, the task is to determine is the two handwritings are of the same writer or not. This problem is formulated in terms of machine learning as 'map a set of input features x to a real-valued scalar target y(x,w).'

1.1. Datasets:

Two datasets based on the extracted features for the sample 'AND' were provided:

1. Human observed dataset
2. GSC Dataset

The human observed dataset has 9 features and the GSC dataset has 512 features for every image of the sample. The target values for all pairs of the samples are given as a measure to train the machine learning models. Regression was to be performed under two settings – feature concatenation and feature subtraction for both of the datasets. So, a total of 4 datasets were generated,

1. Human observed concatenated: 18 features for each pair of images
2. Human observed subtracted: 9 features for each pair of images
3. GSC concatenated: 1024 features for each pair of images
4. GSC subtracted: 512 features for each pair of images

1.2. Splitting the datasets into training, validation and testing:

The models are trained on 80% of each datasets, 10% of the remaining data is used for validation and adjusting the weights generated in the training phase of the model and finally the remaining 10% of the dataset is used for testing and calculating the accuracy of the model.

2. Implementation

2.1. Multiple linear regression:

Multiple linear regression attempts to model the relationship between two or more feature variables and a target variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. The regression equation for n variables x_1, x_2, ... , x_n is defined to be

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

where, $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of regression. They can also be called as the weights that are generated after performing linear regression on the training dataset and adjusted after performing linear regression on the validation dataset to improve accuracy.

For performing linear regression two functions are used:

i. the cost function
ii. gradient descent

i. **a cost function** maps values of one or more variables onto a real number. In this case, the event we are finding the cost of is the difference between estimated values, or the difference between the hypothesis and the real values—the actual data we are trying to fit a line to.
The cost function is calculated by calculating the mean squared error using the following formula:

$$\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Where, $h_\theta(x)$ is the predicted value of the target given the value of features X and y is the expected value of the target. Our aim is to minimize the cost function i.e. to minimize the difference between the predicted output and the expected output. This function is used to find the best fit of all the hypothesis.

ii. We can use **gradient descent** to find the minimized value automatically, without trying a bunch of hypotheses one by one. Using the cost function in in conjunction with gradient descent is called linear regression. The initial costs are calculated using the cost function and then adjusted to minimize the loss using the gradient descent algorithm.

The python code for the loss function and gradient descent is as follows:

```
#gradient descen
alpha = 0.0001
iterations=1000

def cost_function(X, Y, B):
    m = len(Y)
    J = np.sum((X.dot(B) - Y) ** 2)/(2 * m)
    return J
inital_cost = cost_function(X, Y, B)
print("Cost before gradient descent: ",inital_cost,'\n')

def gradient_descent(X, Y, B, alpha, iterations):
    cost_history = [0] * iterations
    m = len(Y)

    for iteration in range(iterations):
        # Hypothesis Values
        h = X.dot(B)
        # Difference b/w Hypothesis and Actual Y
        loss = h - Y
        # Gradient Calculation
        gradient = X.T.dot(loss) / m
        # Changing Values of B using Gradient
        B = B - alpha * gradient
        # New Cost Value
        cost = cost_function(X, Y, B)
        cost_history[iteration] = cost
    return B, cost_history

#performing gd
newB, cost_history = gradient_descent(X, Y, B, alpha, iterations)
print("Co-efficients of regression: ",newB, '\n')
print("Cost after gradient descent: ",cost_history[-1], '\n')
```

X is the feature vector, Y is the output vector and B contains the coefficients of regression that were initialized to 0 before performing. gradient descent.

The following observations were made after performing linear regression on the Human observed dataset:
After gradient descent, the concatenated ERMS is 0.49707653960045006 and the subtracted ERMS is 0.5003009551585283

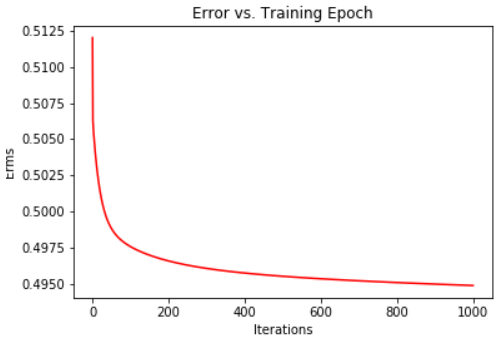


Fig. 2.1.1. Concatenated

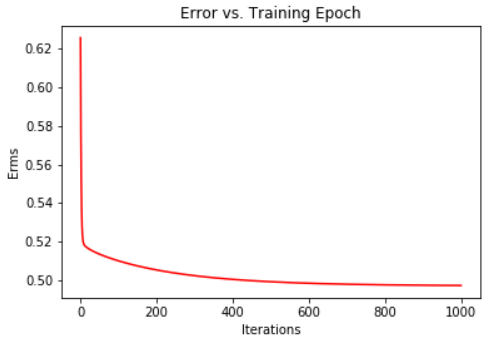


Fig. 2.1.2. Subtracted

The following observations were made after performing linear regression on the GSC dataset:
After gradient descent, the concatenated ERMS is 0.515747855960503 and the subtracted ERMS. is 0.42229832855960503

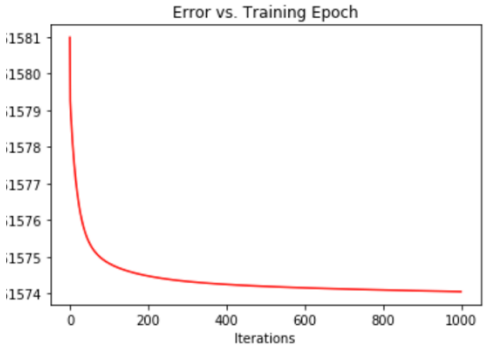


Fig. 2.1.3. Concatenated

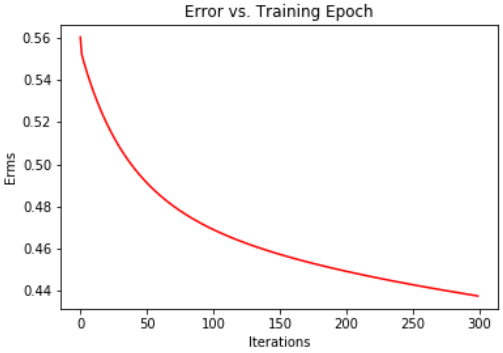


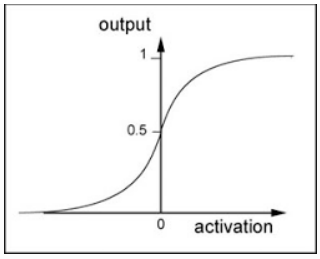
Fig. 2.1.4. Subtracted

2.2. Logistic Regression:

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.

Sigmoid Activation:

In order to map predicted values to probabilities, we use the sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities. Below is a sigmoid curve and the sigmoid equation,



$$f(x) = \frac{1}{1+e^{-(x)}}$$

The code of logistic regression is similar to the one for linear regression with the following changes:

i. The cost function in linear regression is replaced by the following code for logistic regression:

```
def cost_function(features, labels, weights):
    observations = len(labels)
    predictions = predict(features, weights)

    # error when label=1
    class1_cost = -labels*np.log(predictions)

    # error when label=0
    class2_cost = (1-labels)*np.log(1-predictions)

    #Take the sum of both costs
    cost = class1_cost - class2_cost

    #Take the average cost
    cost = cost.sum()/observations

    return cost
```

ii. The predicted output is passed through the sigmoid function to determine the classes of the samples. A decision boundary is used to map the output into discrete classes. A threshold is decided and above which we will classify values into class 1 and below which we classify the values into class 2. Here, we consider the threshold to be 0.5.

$p \geq 0.5$, class=1
$p < 0.5$, class=0

2.3. Neural Network:

A two-layer neural network was created. The hidden layer had 512 nodes and the output layer has 2 nodes for the two values of the target (1=writer match, 0=writer mismatch). The input layer contains 18 nodes for the concatenated datasets and 9 nodes for the subtracted datasets.
The hyperparameters that were considered are batch size = 32, optimizer = rmsprop, loss= categorical_crossentropy.

The following observations were made after training the model on the Human observed dataset:

i. Concatenated: (for 100 epochs)

```
Epoch 1/100
1265/1265 [==============================] - 0s 137us/step - loss: 0.7498 - acc: 0.5130
- val_loss: 0.7052 - val_acc: 0.4984

Epoch 100/100
1265/1265 [==============================] - 0s 14us/step - loss: 0.4686 - acc: 0.7842
- val_loss: 0.7951 - val_acc: 0.5426
```

The training accuracy increased from 0.5130 to 0.7842 after 100 epochs. The validation accuracy increased from 0.4984 to 0.5426 after 100 epochs. The graphs for training accuracy and validation accuracy vs. training epochs are given in fig. 2.3.1.

ii. **Subtracted:** (for 100 epochs)

```
Epoch 1/100
1265/1265 [==============================] - 0s 131us/step - loss: 0.7131 - acc: 0.5043
- val_loss: 0.6913 - val_acc: 0.5237
Epoch 100/100
1265/1265 [==============================] - 0s 10us/step - loss: 0.6076 - acc: 0.6767
- val_loss: 0.7040 - val_acc: 0.5457
```

The training accuracy increased from 0.5043 to 0.6767 after 100 epochs. The validation accuracy increased from 0.5237 to 0.5457 after 100 epochs. The graphs for training accuracy and validation accuracy vs. training epochs are given in fig. 2.3.2.

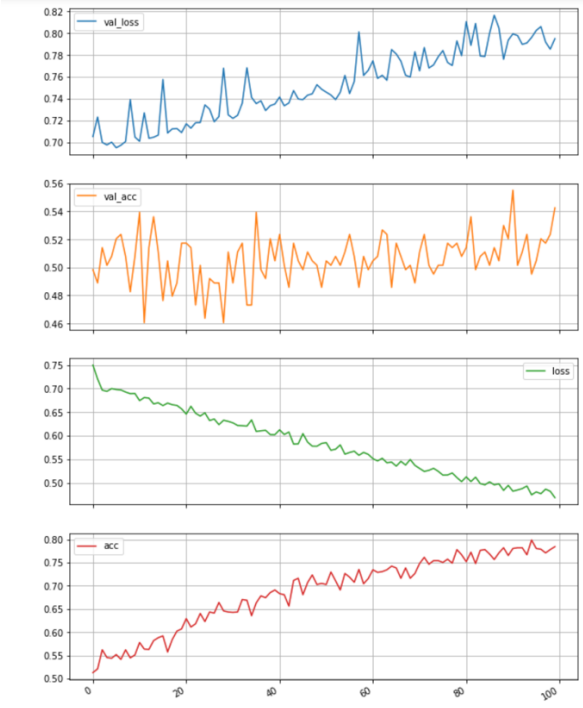


Fig. 2.3.1.

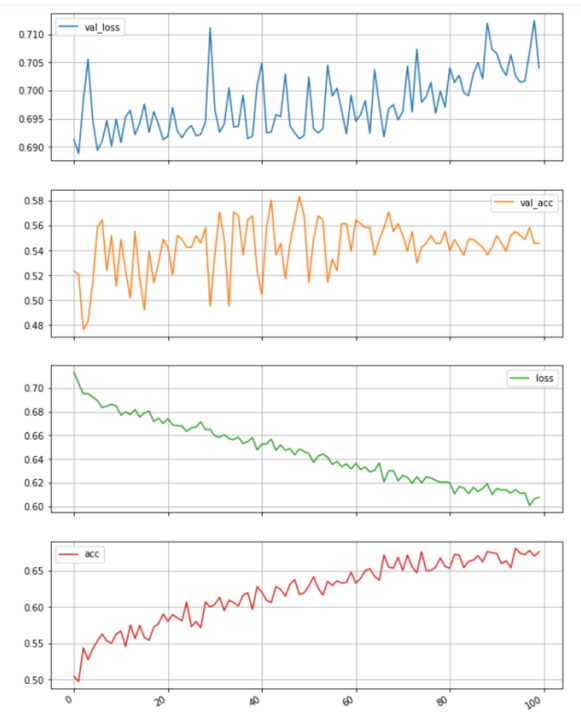


Fig. 2.3.2.

The following observations were made after training the model on the Human observed dataset:

iii. **Concatenated:** (for 1000 epochs)

```
Epoch 1/1000
80000/80000 [==============================] - 1s 10us/step - loss: 0.6886 - acc: 0.532
7 - val_loss: 0.6871 - val_acc: 0.5377
Epoch 355/1000
80000/80000 [==============================] - 1s 8us/step - loss: 0.6426 - acc: 0.5773
- val_loss: 0.6430 - val_acc: 0.5775
Epoch 00355: early stopping
```

The training accuracy increased from 0.5327 to 0.6871 after 355 epochs. The validation accuracy increased from 0.5377 to 0.5775 after 355 epochs. The graphs for training accuracy and validation accuracy vs. training epochs are given in fig. 2.3.3.

iv. **Subtracted:** (for 1000 epochs)

```
Epoch 1/1000
80000/80000 [==============================] - 1s 12us/step - loss: 0.6845 - acc: 0.548
2 - val_loss: 0.6827 - val_acc: 0.5534
Epoch 134/1000
80000/80000 [==============================] - 1s 10us/step - loss: 0.6811 - acc: 0.554
6 - val_loss: 0.6823 - val_acc: 0.5552
Epoch 00134: early stopping
```

The training accuracy increased from 0.5482 to 0.5546 after 134 epochs. The validation accuracy increased from 0.5534 to 0.5552 after 134 epochs. The graphs for training accuracy and validation accuracy vs. training epochs are given in fig. 2.3.4.

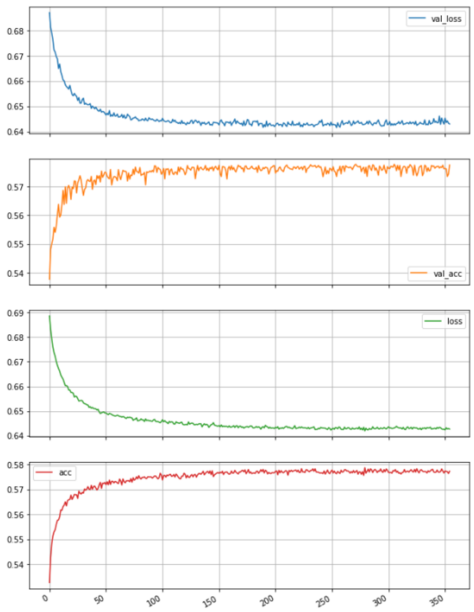


Fig. 2.3.3.

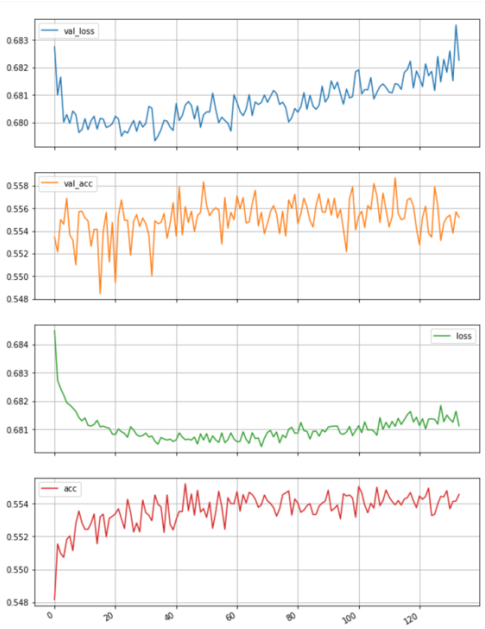


Fig. 2.3.4.

3. Conclusion

Regression was performed on the four given datasets. Linear regression is not a good machine learning approach for this dataset as the targets are discrete binary classes. Logistic regression performs the best under such conditions. The GSC dataset contains more features and thus, gives more accurate machine learning models. The deep learning approach is an altogether better approach for any machine learning applications.

4. References

i. http://www.stat.yale.edu/Courses/1997-98/101/linmult.html
ii. https://medium.com/@lachlanmiller_52885/understanding-and-calculating-the-cost-function-for-linear-regression-39b8a3519fcb
iii. https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html