PROJECT REPORT

CLASSIFICATION ON PIMA INDIANS DIABETES DATASET

SOUMIL GHOSH

ID – 50400736

EMAIL – SOUMILGH@BUFFALO.EDU

Introduction

Our task in this project was to perform Classification on the PIMA Indians Diabetes Dataset to predict whether a person with the same diagnostic measurements as the sample data can have diabetes or not using Logistic Regression and Neural Networks. We have used multiple feature Logistic Regression and Neural Networks with different regularization techniques to solve our problem.

Dataset

We have used the PIMA Indians Diabetes dataset to predict diabetes. The PIMA Indians dataset has been prepared by collecting medical data from female patients above the age of 21 and contains 768 samples of data. Each of the samples has eight features which have been used for training and testing purposes.

These 8 features include:

- Blood Glucose Level
- Number of Pregnancies the patient has had
- Blood Pressure (in mm Hg)
- Triceps Skin Fold Thickness (in mm)
- Insulin Level
- Body Mass Index $\frac{Weight(in\ kg)}{Height(in\ m)^2}$
- Diabetes Pedigree Function
- Age (in years)

Data Pre-processing

Data Extraction

The entire dataset in .CSV format is first loaded into a Pandas dataframe. The "Outcome" field is then dropped to create a dataframe for the Target data. So, we now have separated the dependent and independent variables.

Data Partitioning

The dependent and independent data is then split into Training, Testing and Validation data in the ratio $60:20:20$ respectively. We have used $sklearns.train\_test\_split$ for this purpose. First, we have separated the test dataset using a $80:20$ split using a $random\_state$ of 5 and then we have done a $75:25$ split on the remaining part to get the train and validation datasets.

Methodology

For detecting diabetes, we have used two techniques:

- Logistic Regression
- Neural Networks with L1, L2 and Dropout Regularization

Logistic Regression

Sigmoid Function

We have used sigmoid function for Logistic Regression which is expressed as

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{where z is expressed as}$$

$$z = \sum_{i=0}^{n} w_i \, . \, x_i$$

Here w is basically the weights for each of the features. As there are 8 features for us to consider so there will be a total of 8 weights.

Gradient Descent

We have used Gradient Descent to optimize the value of our weight vectors. This is calculated as follows:

$$dz = X_{Train}^{Transpose} . (Y_{Predicted} - Y_{Actual})$$

$$w = w - \alpha . dz$$

Here dz is the deviation of z or change in z, $X_{Train}^{Transpose}$ is the Transpose of the independent variable matrix which is multiplied by the difference matrix between the predicted value of the dependent variable and the actual value.

After this the weights are updated by subtracting dz times learning rate from the previous weight value. Here α is the Learning Rate.

Neural Networks

A Neural Network is basically constructed from three types of layers:

- Input Layer – This layer feeds the input data into the hidden layers. The input layer normally contains the same number of nodes or neurons as the number of features, but it sometimes contains an extra node which represents the bias.

- Hidden Layers – Each hidden layer takes the output of its previous layer as its input sends its output to the next layer. The hidden layers contain various activation functions which are used for computation. There are various types of activation functions which can be used like "Sigmoid", "Tanh", "RelU" etc. The Regularization is also performed in these hidden layers. In our project we have used 2 regularization techniques, L1-L2 Regularization and Dropout Regularization.

- Output Layer – The output layer normally contains only one node or neuron. This layer contains activation functions like "SoftMax" or "sigmoid" which give the final output

Regularization in Neural Networks

Normally when there are many neurons or nodes and in case of multilayer neural networks, overfitting is a very common scenario. To avoid this, regularization is used, which operates by constraining the range of values of the weights of a neural network. In our project we have demonstrated the operation of two regularization techniques:

- L1-L2 Regularization
  1. In case of L1 Regularization, the absolute value of the weights which are very small or close to zero are made equal to zero by the regularizer.
  2. In case of L2 regularization, both small weights and large weights are transformed into a number close to zero but not quite zero.

- Dropout Regularization – Dropout Regularization works by deactivating a part of the hidden layer. It means that the contribution of a group of randomly selected neuron of the hidden layer is ignored by the model during training.

Experimental Setup

Logistic Regression

We have trained our model for *10000 epochs,* and we have assumed the learning rate to be 10^{-6}. The size of our feature matrix is $(460{,}8)$, so we have taken a weights matrix of $(8{,}1)$. First, we have initialized our weights matrix by 1 and the multiplied the two to get the z matrix using the formula:

$$z = X_{Train}w$$

After this we have calculate the gradient or deviation in z using the formula:

$$dz = X_{Train}^{Transpose}.(Y_{Predicted} - Y_{Actual})$$

Here dz is the gradient matrix which we use to update our weights using the formula:

$$w = w - \alpha.dz$$

Where α is the learning rate.

After the training is completed, we test our model on the validation and test dataset by using the sigmoid function. If the sigmoid function output is greater than 0.5, we update it as 1 “diabetes positive” else update it as 0 “diabetes negative.”

Then we compare our predicted test and validation outputs to the actual ones to get the accuracy.

We also import $sklearns.metrics$ library to calculate metrics like Precision, Recall, F1-score ROC AUC Score, Cohen’s Cuppa and Confusion Matrix to evaluate our model.

Neural Networks

In our project, we have used TensorFlow to design our neural network model.

We have designed a model with three hidden layers which contain $40, 25\ and\ 15$ neurons respectively. In these hidden layers we have used “RelU” activation function and specified the input dimension as 8 as we have 8 features in our dataset.

We have experimented our model using two regularization techniques namely L1-L2 Regularization and Dropout. We have used L2 Regularization of 0.1 in one model and Dropout of 0.2 in a separate instance of the same model to compare the impact of different regularization techniques.

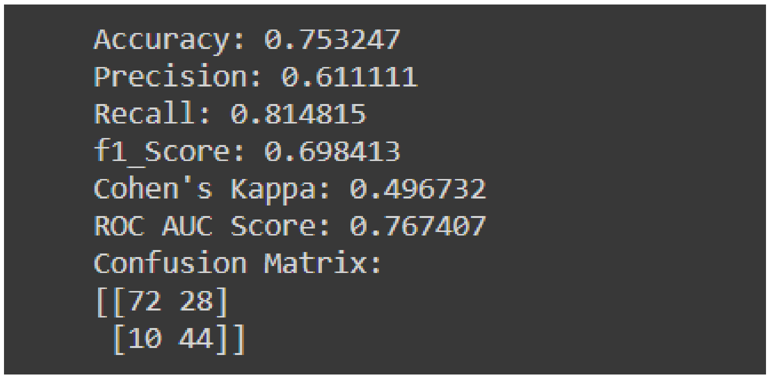
In our output layer we have used “sigmoid” activation function. We have used “Adam” optimizer along with a learning rate of 10^{-5}. For loss evaluation we have used “binary_crossentropy” loss function.

We trained our model for 500 epochs with a batch size of 8.

Results

Logistic Regression

After training our model for 10000 $epochs$ using a learning rate of 10^{-6}, we receive an accuracy of 75.32% on the test dataset. The other parameter calculated are indicated in the image below.



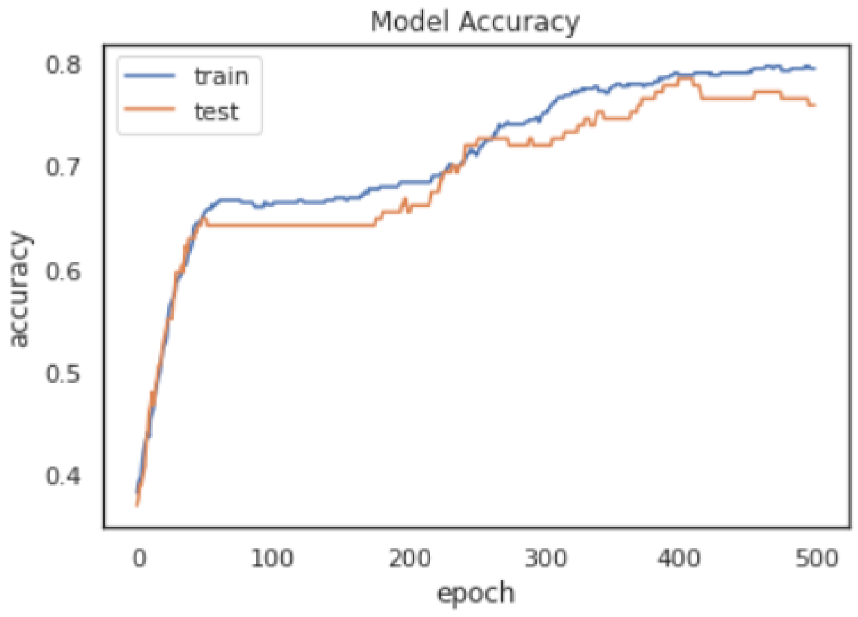
```
Accuracy: 0.753247
Precision: 0.611111
Recall: 0.814815
f1_Score: 0.698413
Cohen's Kappa: 0.496732
ROC AUC Score: 0.767407
Confusion Matrix:
[[72 28]
 [10 44]]
```

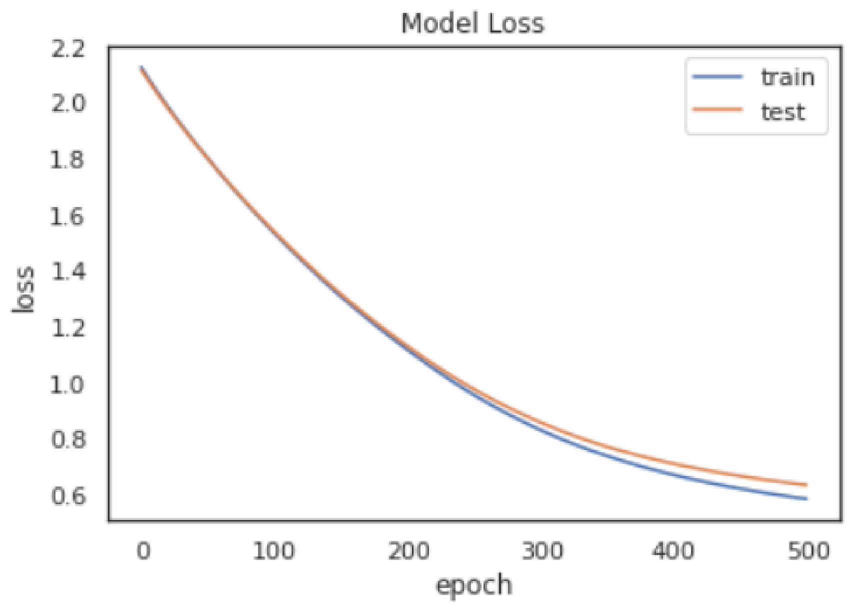
Neural Networks

L1-L2 Regularization

After training our model for 500 $epochs$ with a learning rate of 10^{-5} our model achieved an accuracy of 80.52% with a loss of 0.5759, when we used L2 regularization of 0.1.

The loss and accuracy curves for L1-L2 Regularization are shown below:

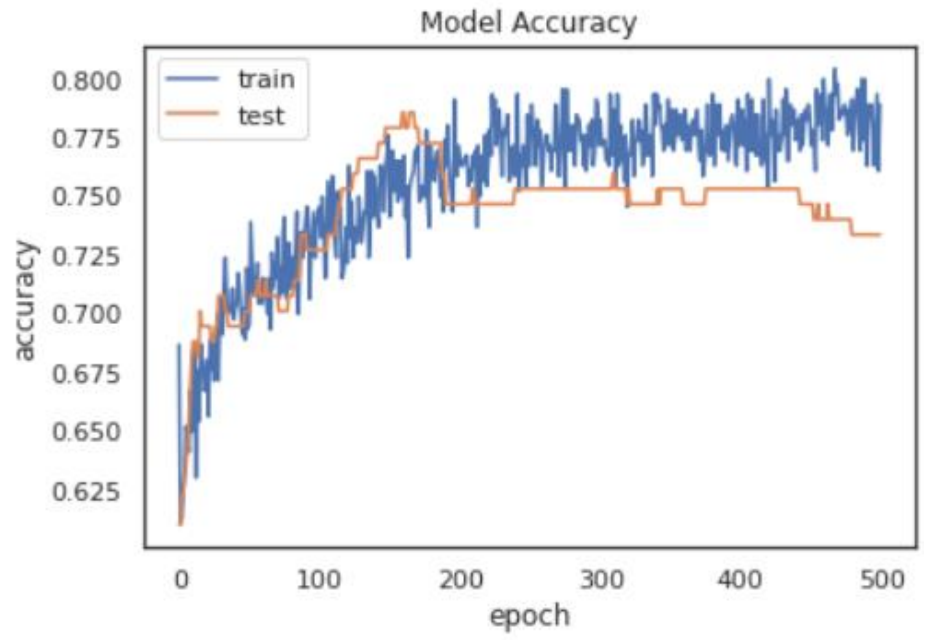


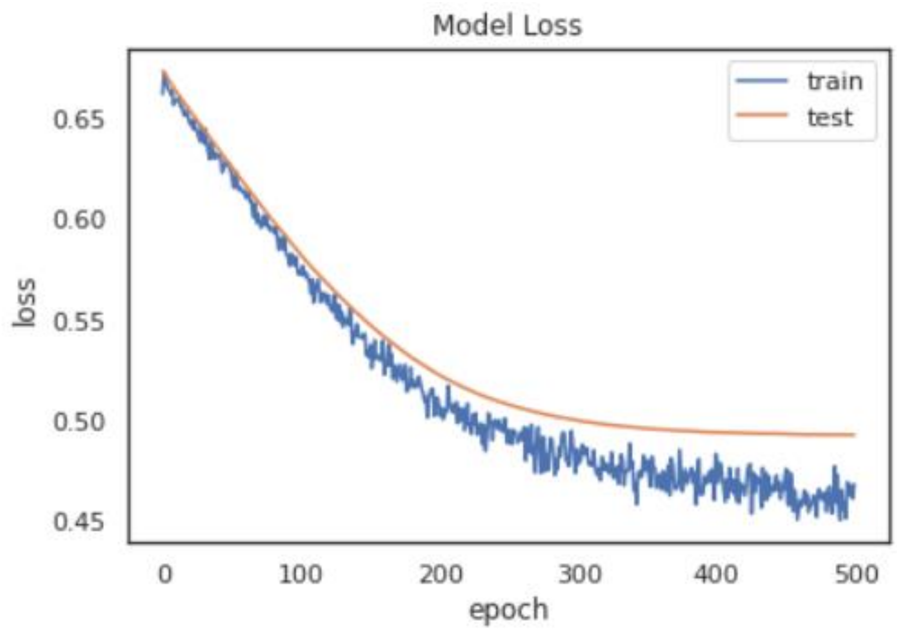


Dropout Regularization

After training our model for 500 epochs with a learning rate of 10^{-5}, our model achieved a prediction accuracy of 76.62% with a loss of 0.4536 when we used Dropout Regularization of 0.2.

The loss and accuracy curves for Dropout Regularization are shown below:





From the graphs, it is observed that the validation accuracy in case of Dropout decreases after approximately $200\, epochs$ whereas the accuracy of L2 increases till $500\, epochs$ almost. Therefore, overfitting occurs for dropout after $150\, epochs$. From here we can say that Dropout converges faster than L2 when both are applied to only one hidden layer.

From the accuracy it is observed that L2 yields much higher prediction accuracy than dropout when both are applied to only one hidden layer.

Comparing Neural Networks with Logistic Regression, it is observed that the prediction accuracy for Neural Networks (80.52) is much higher than that compared to Logistic Regression (75.32). Moreover, Logistic Regression required much greater number of epochs to converge compared to neural networks.

Conclusion

Logistic Regression is definitely a very good classifier, and it was able to classify the probability of whether a person has diabetes or not quite well, but the prediction accuracy improved quite a bit when we used Neural Networks. The number of epochs required for neural networks was also much less compared to logistic regression which required 10000 epochs for convergence.

This main difference is caused because a neural network is much more robust than logistic regression. Logistic Regression can be thought of a neural network model with just one hidden node and one activation function. Neural Networks, instead, combine the impact of different hidden nodes and different activation functions, so the training and prediction is much better in case of neural networks.

In case of regularization, it is observed that L2 regularization yields better prediction accuracy than dropout. Therefore, we can say that when there are small number of neurons in the hidden layers L2 regularization can yield better results than dropout. Furthermore, as the number of hidden layers increases, the accuracy of dropout is also seen to decrease.