Assignment 1 - Final

Sai Mahesh Pullagura
10/10/2021

Overview

The task of this project is to perform classification using Machine Learning. With the help of the dataset "Pima Indians Diabetes Database dataset", a patient has been classified if he/she has diabetes (1) or not (0) using logistic regression and neural network as the classifier and the code has been implemented using Python.

Dataset

The provided dataset contains medical data of female patients with the diagnostic measurements of 8 features and a total of 768 instances of input data.

Data processing

- I have extracted the given data using the function '**read_csv()**' that is present under the in-built module 'pandas' and then segregated the complete data into 2 parts: X that consists of rows with all 8 features and Y with the actual classification value (0 or 1).
- I have then partitioned the obtained data using '**train_test_split()**' under the module 'sklearn'. I have used the function twice to split the data into 3 parts: 60% of training data, 20% of Validation data and the rest for testing.
- In order to normalize the data before proceeding with further steps, the function "**fit_transform()**" comes into picture which normalizes the training input data and the function '**transform()**' to normalize test and validation datasets.

Part I -- Logistic Regression

- The main idea behind Logistic Regression is to train the model so that it could correctly classify data given the feature vectors. In order to achieve this, we make use of gradient descent to bring the error rate to minimum.
- To begin with, let us consider we have an input vector **X** that is characterized by **m** parameters. We have the actual classification **Y** for all such input dataset. We first begin with assigning default **weights** to each of the parameter and also add a constant **bias** to control the weight updation activity.
- Then with the help of sigmoid function and error function, we iteratively update the weights and bias till the error reaches to a global minimum value. Once the error reaches to a constant (after multiple iterations), we then believe the model is now ready for classification and test it with the help of validation and testing datasets.

- **Programmatically**, we first define the class '**LogReg**' and initialize the parameters *Learningrate* and *Iterations*(epochs) to 0.01 and 100 respectively.
- Secondly, we implement the function '**TrainData()'** to update the values of weight and bias accordingly.
- For each iteration, we first find the sigmoid value of the input function. The input function is represented as a product of input with it's corresponding weight added by a constant bias b. The sigmoid of this function is given by :

  p=1 / (1 + np.exp(-z)), where z= wT.x+b

- Once p is calculated, we then determine the cost, which is given by:

  L= - ∑ (y logp + (1-y) (1-logp))
  1->m

- Now in order to update the weights, we find the gradient descent using partial derivatives of dz, dw and db. They are given by:

  dz=p-Y, where p is the sigmoid value and Y is the actual target.
  dw = (1 / m) * np.dot(X, dz.T)
  db = (1 / m) * np.sum(dz)

- We then update the weights and bias values as below:
  w = w - Learningrate * dw
  b = b - Learningrate * db

- This process is followed till loss L becomes minimum. Here we are iterating for 100 times and the loss becomes minimal as the no of iterations pass by.

- Once the model is ready to classify, we validate it using the test and validation data sets. For this purpose, we define a function 'predict()' that will perform sigmoid operation and output the value 1 if sigmoid is greater than 0.5, else the output will be 0. It is given by:

  Y_pred=np.where(sigma>0.5,1,0)

- Once we obtain the predicted output values, we then try to find the accuracy of the model. If the actual output and the predicted out put is the same, then we have a hit, else we got a mismatch. Sing this data, we find both accuracy and it's corresponding error rate as follows:

  Accuracy = (No of correctly classified data / total input size) * 100
  Error_rate = (No of incorrectly classified data / total input size) * 100

Results – Logistic Regression

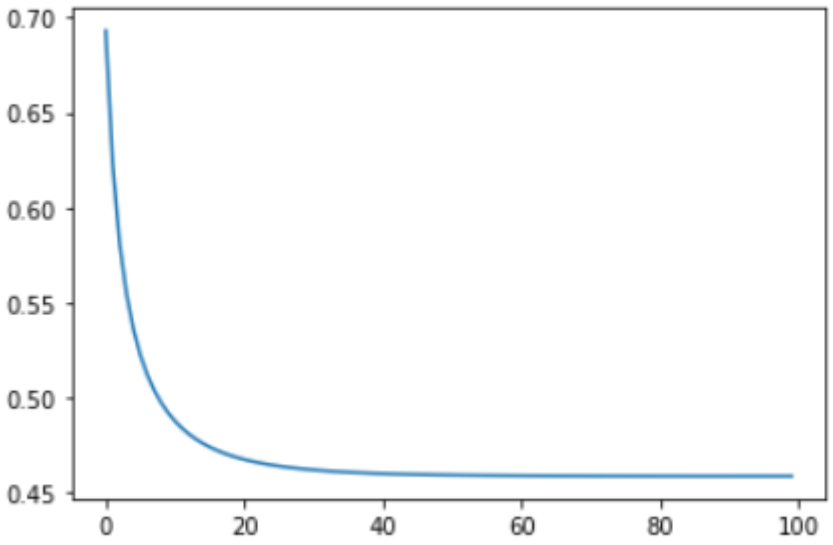
- For the above implemented approach, we have obtained accuracy and error_rate as below:

```
Accuracy achieved by the model on Test set :  77.92 %
Error rate determined by the model on Test set :  22.08 %

Accuracy achieved by the model on Val set :  75.97 %
Error rate determined by the model on Val set :  24.03 %
```

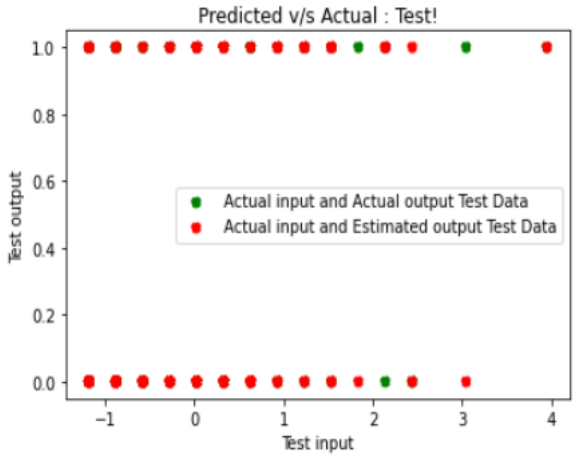
- The error or the loss value could be seen decreasing and it reaches to the global minimum value as we go through a no of iterations. The graphical representation of the same could be seen as below:

```
Error plot is obtained as below.
```

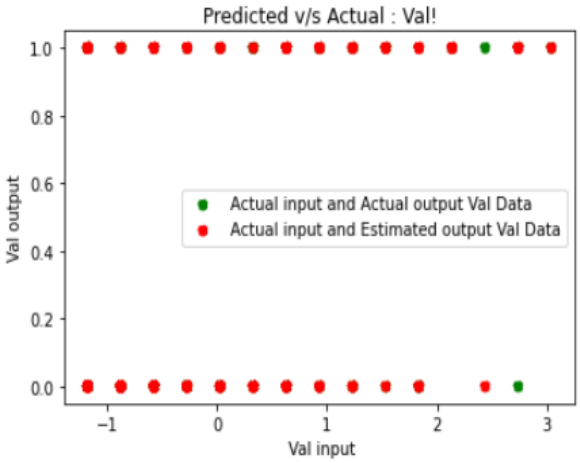


- I have also plotted a Graph comparing the actual and estimated output values for both test and validation input data as seen below.

```
Graph comparing the actual and estimated output values for test input data is shown below.
```




```
Graph comparing the actual and estimated output values for Val input data is shown below.
```



Part II -- Neural Network

- The main usage of Neural Network is similar to that of Logistic Regression whose functionality is to train the model so that it could correctly classify the data given feature vectors.
- Neural network transforms the given input to a non-linear form with the help of **activation function** (relu, sigmoid etc) and the corresponding weights are updated using an **optimizer** (like SGD , adam etc). However, the main difference between the two is that Neural network comprises of **multiple hidden layers** between the input and the actual output which in-turn will be performing similar transformations on the input it receives and reaches the final output layer.
- Once the actual output layer is reached, we determine the **loss function**, and then back propagate the same through all the hidden layers to update the corresponding weights. The process continues iteratively (for the selected epoch value) till we get an optimal solution.

- **Programatically**, we will be implementing neural networks with the help of **Regularization technique**, which is used in prevention of over-fitting of data.
- Post loading the given dataset, we will first define the hidden layers that will be implicated in the model. We are creating a **3-layer model** (2 hidden layers and 1 output layer) with the help of **model.add()** from **keras** module.
- While defining the hidden layer, the activation and regularization parameters are indicated. Note that we are making use of "**relu**" activation function in the 2 hidden layers , "**Sigmoid**" activation function in case of output layer (as this is a binary classification problem) and the regularization used here is **l1**.

- A sample hidden layer is given by :
  model.add(Dense(400,activation='relu',activity_regularizer=keras.regularizers.l1(0.01)))
  Here:
  Dense() makes sure that each neuron in the dense layer receives input from all neurons of its previous layer.
  400 indicates the no of neurons in that particular layer.
  The other parameters: Activation and activity_regularizer are used to select the activity function and regularization technique respectively.

- Next, we define the **loss**, **optimizer** and **metric** parameters with the use of **compile()**. The loss function used is '**binary_crossentropy**' as it is a 2-class problem., the optimizer selected is stochastic gradient descent (**sgd**) and the metric that we are looking to find is **accuracy**.

- It is given by:
  model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

- Now the model is trained using **fit()** function that takes the training and validation data as input.
  model.fit(X_Train, Y_Train, validation_data = (X_Val,Y_Val), epochs=100, batch_size=30)

- Finally once the model is trained such that the error rate is minimized, we then test the model using '**evaluate()**' which will return the accuracy and loss values for that given data.
  Val_loss, Val_accuracy = model.evaluate(X_Val, Y_Val).

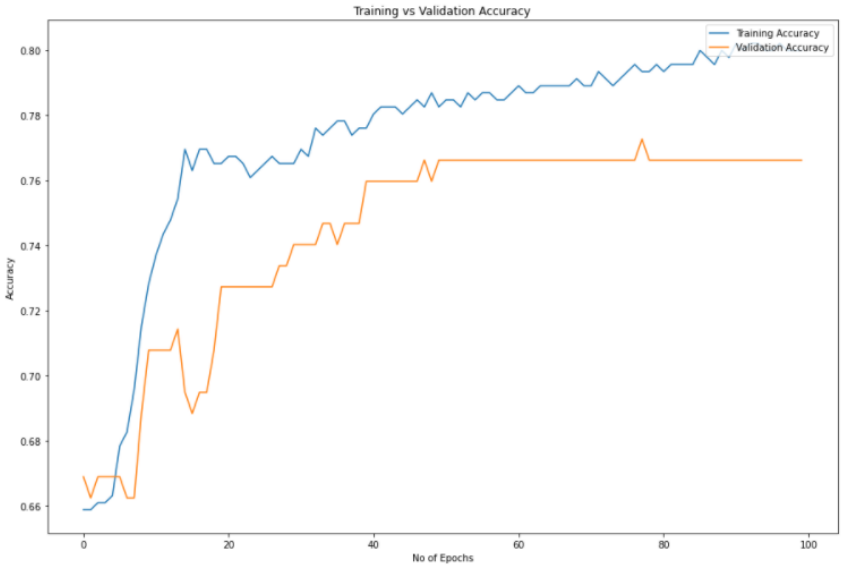
Results – Neural Networks using L1 Regularization

- For the above implemented approach, we have obtained accuracy and error_rate as below:

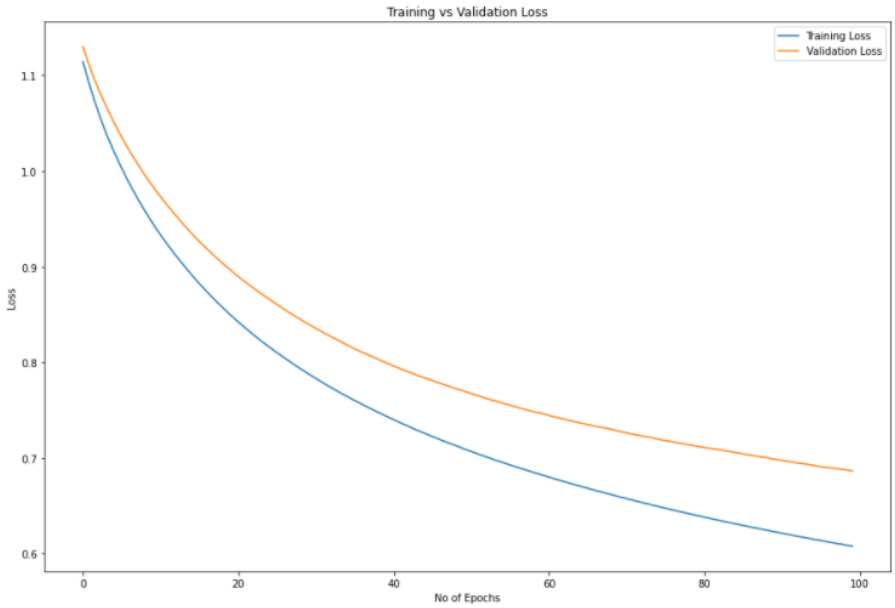
```
Accuracy achieved by the Neural Network model on Test set :  80.52 %
Error rate determined by the Neural Network model on Test set :  64.72 %

Accuracy achieved by the Neural Network model on Val set :  76.62 %
Error rate determined by the Neural Network model on Val set :  68.66 %
```

- The plot between training and validation accuracy as seen in the Neural Network model is as follows:



- The plot between training and validation loss as seen in the Neural Network model is as follows:



Part III -- Implementing Neural Networks using different Regularization Techniques

- In the final part, the neural network has been implemented using L1 and Dropout Regularization techniques.
- **Dropout** is a technique where randomly selected neurons are ignored during training the model so that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.
- During implementation, the model that uses L1-regularization technique is the same model that was defined in the previous step (Part II).

- For inserting dropout technique, we simply add a dropout layer as below: **model1.add(Dropout(0.01))**

- We then compare both models by determining the accuracy and loss values estimated by each model.

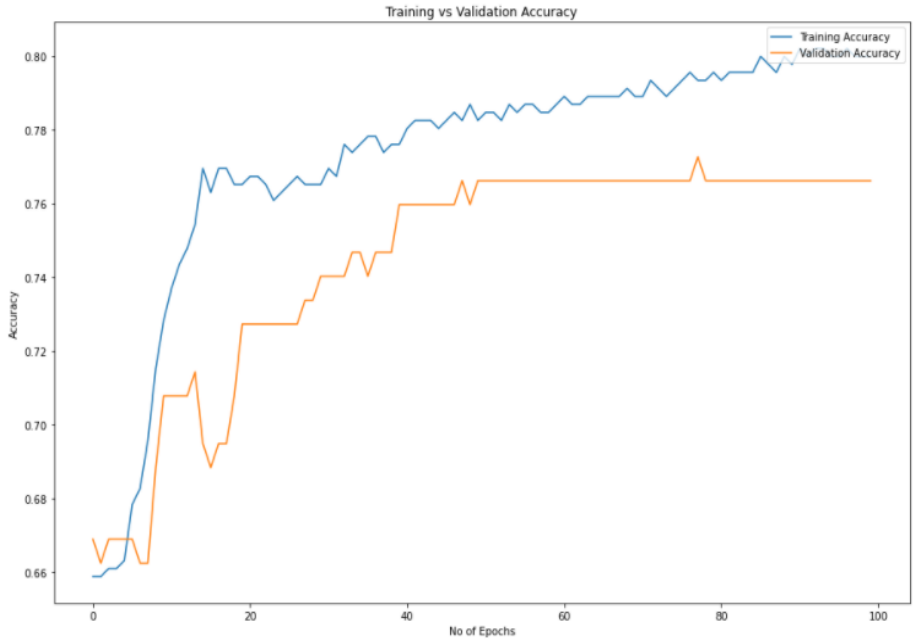
Results – Neural Networks using L1 and Dropout Regularization

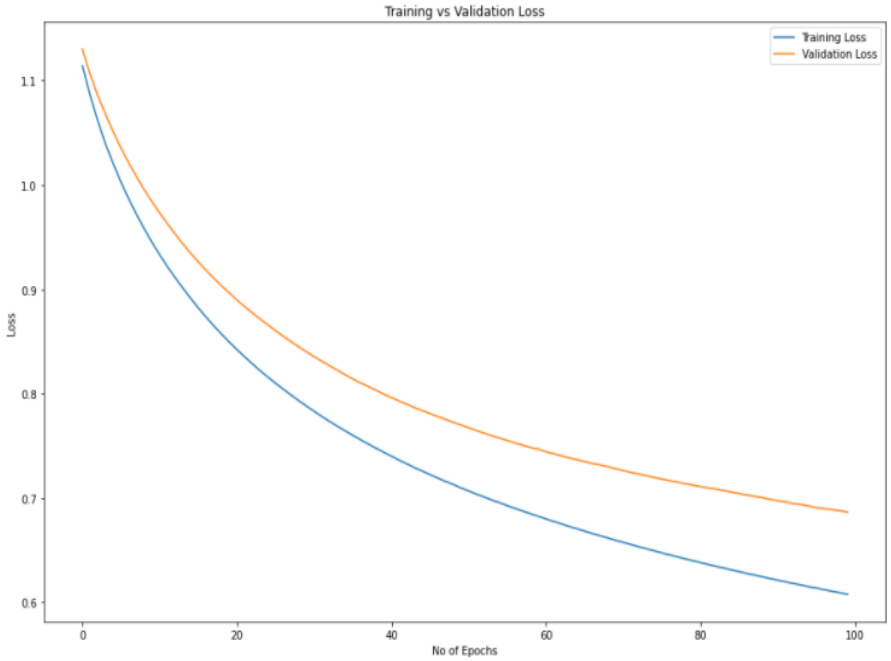
- The Performance Metrics (or the accuracy-error rate) obtained by modelling the neural network using **L1-Regularization technique** is shown as below:

```
Accuracy achieved by the Neural Network model on Test set :  79.22 %
Error rate determined by the Neural Network model on Test set :  64.7 %

Accuracy achieved by the Neural Network model on Val set :  76.62 %
Error rate determined by the Neural Network model on Val set :  69.04 %
```

- The plot between training vs validation Accuracy and loss parameters using **L1-Regularization technique** is seen as follows:



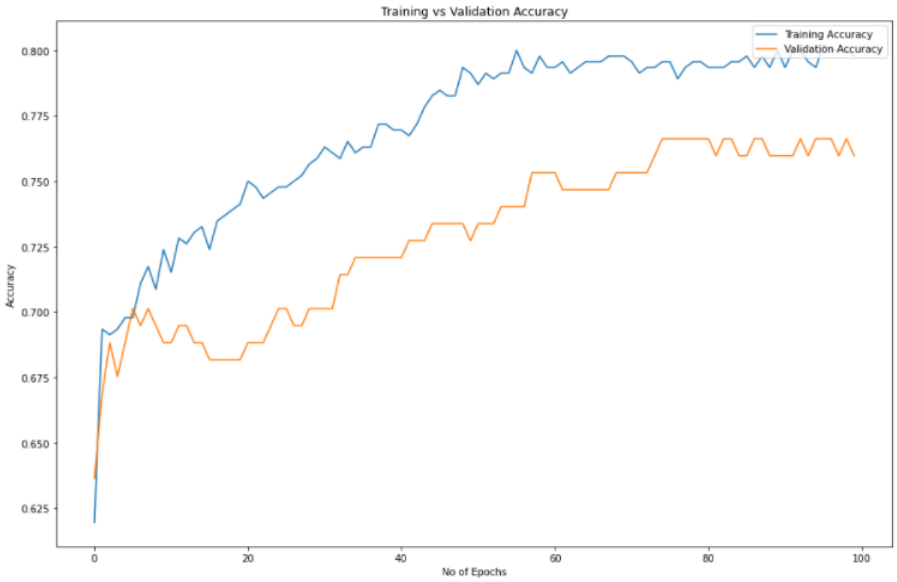


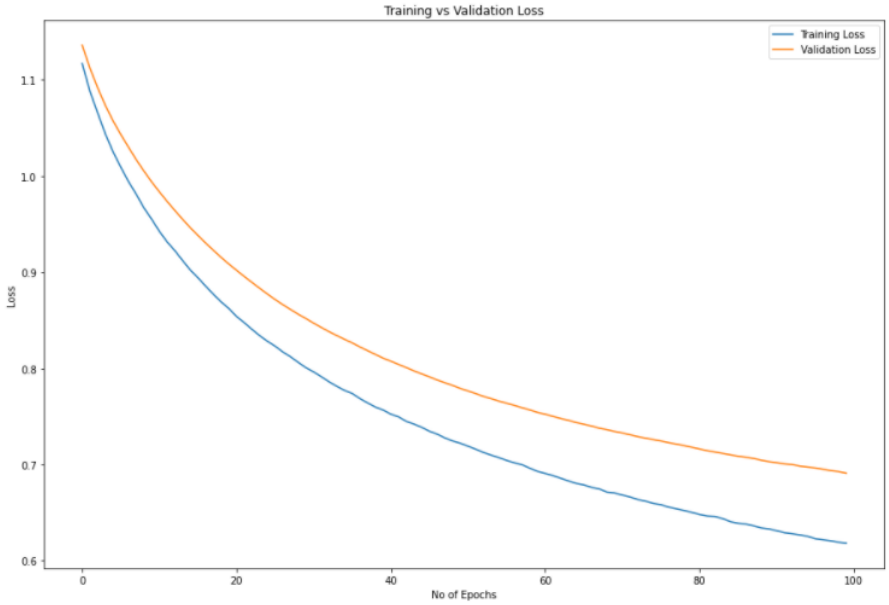
- The Performance Metrics (or the accuracy-error rate) using **Dropout-Regularization technique** is shown as below:

```
Accuracy achieved by the Neural Network model on Test set :  81.82 %
Error rate determined by the Neural Network model on Test set :  64.32 %

Accuracy achieved by the Neural Network model on Val set :  75.97 %
Error rate determined by the Neural Network model on Val set :  69.1 %
```

- The plot between training vs validation Accuracy and loss parameters using **Dropout-Regularization technique** is seen as follows:





- Note that for the above implementation, there is **no significant changes visible** in the accuracy of the model between the 2 regularization techniques.

References

- Lecture slides -- 4.3.2-LogisticReg.pdf
- Chaitra cheluvaraju -- chaitracheluvaraju_assignment_sample_report.pdf
- Soumyya Kanti Datta and Nitin Kulkarni -- http://localhost:8888/notebooks/CSE574%20ML/Professor%20Jupyter%20Notebook/Neural_Networks.ipynb
- Url -- https://www.geeksforgeeks.org/implementation-of-logistic-regression-from-scratch-using-python/
- Url -- https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/.