CSE 574 Introduction to Machine Learning

Project 1

Rahul Reddy Karumuri (50416790)

1.Overview:

The goal of this project is to implement classification using machine learning for a two-class problem and predict weather the patient has diabetes or not. In the first part of this project, I have implemented feature extraction, data processing and trained the model using gradient descent and calculated the accuracy using Logistic Regression. While in the second part, I have implemented neural networks with L1 and L2 as regularization methods. Finally, in the third part, I build neural network with dropout and L2 regularization methods and compared both.

2. Dataset:

Dataset that we need to consider is Pima Indians Diabetes Database dataset. This dataset contains medical record of 768 females with age greater than 21. I have split data samples into training, validation and testing each constituting of 60%, 20% and 20% respectively. Training dataset has 460 samples whereas testing and validation datasets has 154 samples each.

3. Environment:

I have used Jupiter Notebook on Google Collab for implementation of this project.

4. Data Preprocessing:

4.1 Feature Extraction:

Here, I have extracted the features using pandas' data frame.

4.2 Data Partitioning:

I had split the data for training, testing and validation with 60%, 20%, 20% respectively using train_test_split function from sklearn library

5. Part 1: Implementing Logistic Regression

Logistic Regression: It is a machine learning model that uses one or more independent variables and predict the outcome. The output will be either 0 or 1 (binary classification)

5.1 Normalization:

In this dataset we had 8 features namely:

1. Glucose (Blood Glucose level)
2. Pregnancies (The number of pregnancies the patient has had)
3. Blood Pressure (mm Hg)
4. Skin Thickness (Triceps skin fold thickness (mm)
5. Insulin level
6. BMI (Body Mass Index: weight in kg/(height in m)^2)
7. Diabetes Pedigree Function
8. Age (In years)

These feature values may have a wide range of variability. The range of values for every feature may vary and the Machine Learning model may either underestimate or overestimate the importance of the feature. To scale all the values of the features to a single well-defined range, we do normalization. The type of normalization that I had used is mean Normalization. The formula used for calculating this normalization is

$$x' = \frac{x - \mu}{\max(x) - \min(x)}$$

Fig 1: Mean Normalization formula

Here x is the element and μ is the mean of the dataset.

Here is the function I used to calculate the normalization

```
def normalize(X):
  a=X-X.mean()
  b=X.max()-X.min()
  return a/b
```

5.2 Model Building:

For creating the model I have created the function `logistic_Regression(X,Y,epoch,learningRate)` which will take input data, outcome values, epochs and learning rate as parameters, then I normalize the data with the input data.

Now find the sigmoid values for the given input data. Here are the equations I used to find the sigmoid function

Equations :

$$W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_n \end{bmatrix}_{nx1}$$ ..... initialize with zeros

$B = single\ weight/parameter$

$$X = \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}_{nxm}$$

$$Y = \begin{bmatrix} . & . & . & . & . \end{bmatrix}_{1xm}$$

$\sigma = \frac{1}{(1+e^{-x})}$ ..... (sigmoid function)

A = $\sigma(W^T * X + b)$ ..... (probabilistic predictions of shape (1 x m))

Fig 2: Equation for calculating sigmoid function

Gradient Descent

$dW = \frac{\partial COST}{\partial W} = (A - Y) * X^T$ ..... shape (1 x n)

$dB = \frac{\partial COST}{\partial B} = (A - Y)$

$W = W - \alpha * dW^T$

$B = B - \alpha * dB$

Fig 3: Equation for finding the gradient descent

Cost function :

$$cost = -\frac{1}{m}\sum_{i=1}^{m}[y * log(a) + (1-y) * log(1-a)]$$

Fig 3: Equation for calculating cost function

Here, is the function I created to calculate the sigmoid for the given data:

```
def predict(data,w0,w1,w2,w3,w4,w5,w6,w7,w8):
  sigmoidArray=[]
  for x in range(data.shape[0]):
    sigmoidEquation = 1 / (1 + exp(-1*w0 + -1*w1*(data['Pregnancies'].iloc[x]) + -1*w2*(data['Glucose'].iloc[x])+ -1*w3*(data['BloodPressure'].iloc[x])+ -1*
    sigmoidArray.append(sigmoidEquation)
  numpyArray=np.array(sigmoidArray)
  return numpyArray
```

5.3 Validation Accuracy:

For the validation data set I got the accuracy as: `78.57142857142857 %`

5.4 Test Accuracy:

For the testing data set I got the accuracy as: `79.87012987012987 %`

6. Part 2: Implementing Neural Networks

Neural Networks: Neural networks are computing systems that are inspired from the biological neural networks that constitute human brain. Such systems learn to perform tasks by examples, generally without being programmed, with any task-specific rules

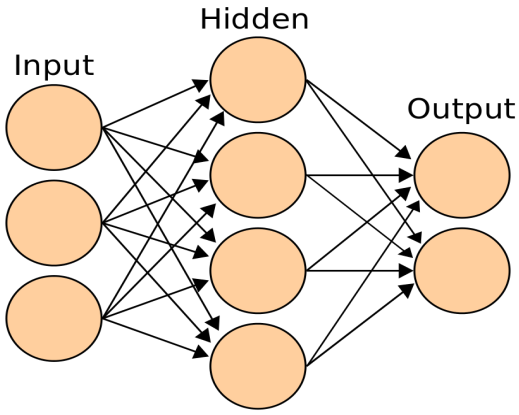


Fig4: Image Neural Networks

6.1 Normalization:

Here, in part 2 for normalizing the data I had used min-max-scalar method from preprocessing module from sklearn library, which will give the values in between 0 to 1

6.2 Regularizations

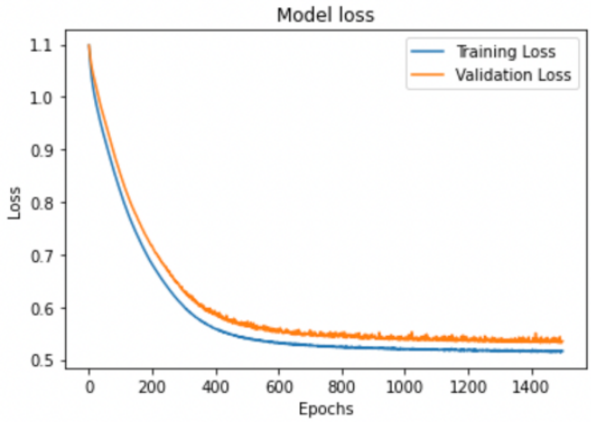
Regularization is a technique which makes slight modifications to the learning algorithm such that the model achieves the better results by avoiding overfitting. This will improve the model performance. In this part I have used L1 and L2 as regularizes

6.3 Building Model

- For building the model I have used 2 hidden layers, for the 1st layer I had given 15 neurons with Rectifier activation function and L1 Regularization method with parameter $\lambda = 0.01$. In the 2nd hidden layer, I had given 12 neurons with Rectifier activation function and L2 Regularization method with parameter $\lambda = 0.01$. Finally, for the output layer I have given 1 neuron with sigmoid activation function.
- Now compile the model using stochastic gradient descent function with learning rate as 0.01. to measure how well our model is performing and, I am passing accuracy as a parameter for finding the accuracy
- Now, train the model by fit method on the training data and with batch size of 32, and with 1500 epochs
- Finally, I have plotted the data with the values, that I got.

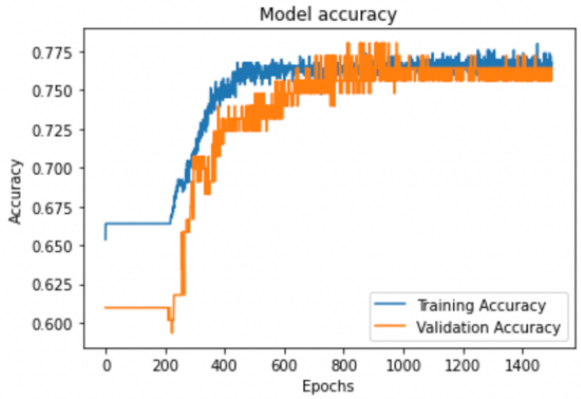
6.4 Results

6.4.1 Training Loss vs Validation Loss



From the graph we can see that model loss decreased drastically for both Training loss and Validation loss before 300 Epochs.

6.4.2 Training Accuracy vs Validation Accuracy



From the graph we can observe that, both training accuracy and validation accuracy increased from 200 Epochs.

6.4.3 Testing Accuracy:

Finally, for the above neural network model I got an accuracy of `77.92207792207793 %` with using L1 and L2 as regularizations

7. Part 3: Implementing different regularization methods for Neural Networks

Here, in part 3 I have built neural network with dropout and L2 as regularizes

7.1 Dropout Regularizer randomly selects some nodes and removes them along with all the incoming and outgoing connections.

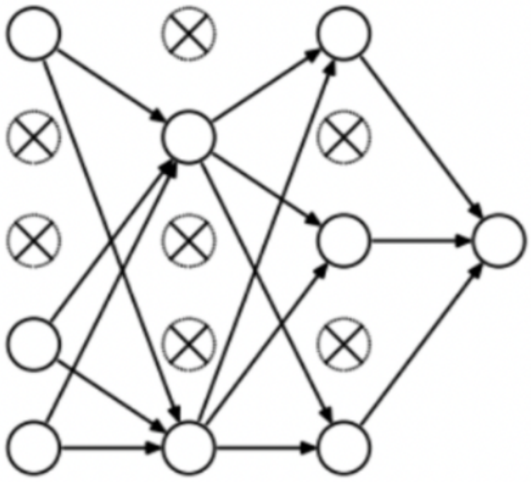


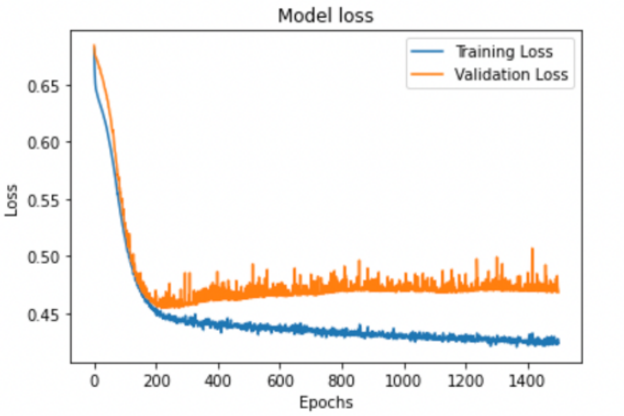
Fig: Dropout regularization example

7.2 Building the model with Dropout Regularizer

- For building the model I have used 2 hidden layers, for the 1st layer I had given 12 neurons with Rectifier activation function and Dropout Regularization method with dropout value 0.4. In the 2nd hidden layer, I had given 15 neurons with Rectifier activation function and dropout Regularization method with dropout value 0.4. Finally, for the output layer I have given 1 neuron with sigmoid activation function.
- Now compile the model using stochastic gradient descent function with learning rate as 0.02 to measure how well our model is performing also, I am passing accuracy as a parameter for finding the accuracy
- Now, train the model by fit method on the training data and with batch size of 32, with 1500 epochs
- Finally, I have plotted the data with the values, that I got.

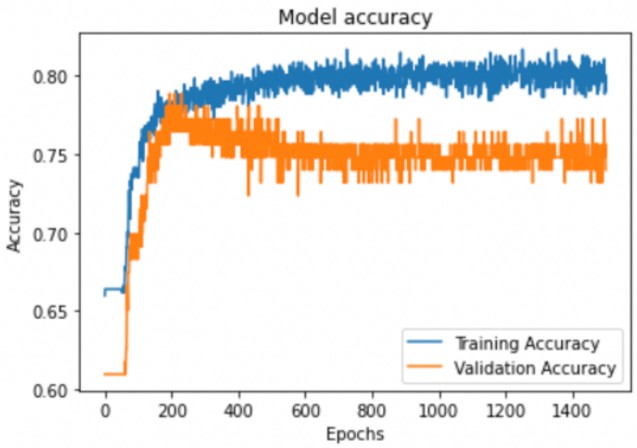
7.2 Results

7.2.1 Training Loss vs Validation Loss



From the graph we can see that model loss decreased drastically for both Training loss and Validation loss before 200 Epochs.

7.2.2 Training Accuracy vs Validation Accuracy



From the graph we can observe that, both training accuracy and validation accuracy increased from 50 Epochs.

7.2.3 Testing Accuracy:

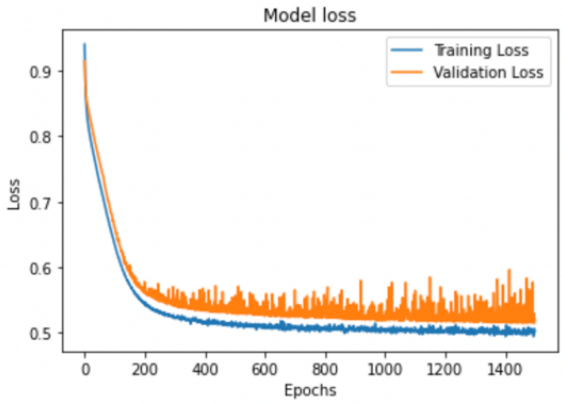
Finally, for the above model I got an accuracy of `75.97402597402598 %` with using dropout as regularizer

7.3 Building the model with L2 Regularizer

- For this model I had used same batch size, learning rate, epochs as that of dropout regularizer, so that we can compare both.

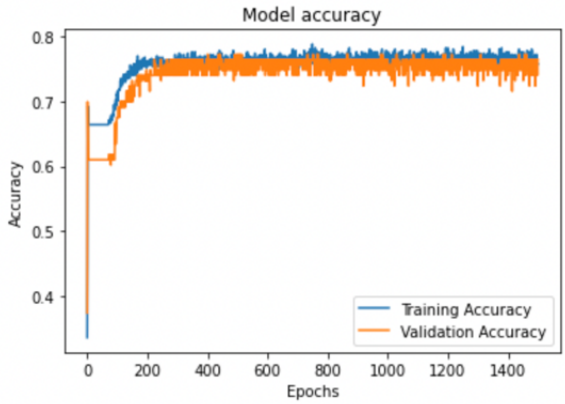
7.3 Results

7.3.1 Training Loss vs Validation Loss



From the graph we can see that model loss decreased drastically for both Training loss and Validation loss before 200 Epochs.

7.3.2 Training Accuracy vs Validation Accuracy



From the graph we can observe that, both training accuracy and validation accuracy increased from starting onwards.

7.3.3 Testing Accuracy:

Finally, for the above model I got an accuracy of `77.27272727272727 %` with using dropout as regularizer

7.4 Comparison

From the above model results we see that, even though both are used to reduce overfitting we can conclude that L2 Regularizations is more efficient than dropout regularizer for the smaller datasets.

8.Conclusion

- We have implemented a logistic regression model developed with gradient descent which gives an accuracy of 79.87%.
- Splitted dataset into 60% training and 20% Validation dataset, helped in tuning the hyper parameters which improved the accuracy of the model.
- Implemented Neural networks with L1, L2 and Dropout regularization methods
- Neural network with L2 regularization method gives better results than the Dropout since we have a smaller dataset and small model.

9.References

1. Build Your Own Artificial Neural Network Using Python (https://randerson112358.medium.com/build-your-own-artificial-neural-network-using-python-f37d16be06bf)

2. Overview of Regularization Techniques in Deep Learning (https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/)