Assignment 1

Gurudev Yalagala

50430743

1. Assignment Overview:

Assignment targets on implementation of linear models for regressions. It is basically 2 parts which concentrates on classification problems. That is to implement Logistic Regression using stochastic gradient descent. Further for the second part, it focuses on implementation of Neural Networks with two hidden layers with different regularization methods (l2, l1). The above implementations need to be executed over the give data set.

2. Dataset:

The dataset given is Pima Indians Diabetes Database to predict the onset of diabetes based on diagnostic measures provided. Data is split into categories for training and testing of 614 for train size and 154 test sizes constituting 80% and 20% of data set.

3. Implementation:

Implementation was done in Google Colab.

3.1. Part 1: Implementation of Logistic Regression.

Logistic Regression is one of the simplest and commonly used ML algorithm approach for two-class classification. It can be employed on baseline models or on any binary classification problem. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables. The following are core concepts and implementation that are applied for training the model.

3.1.1. Implementation

Dataset from csv is taken and implemented # Now since we want the valid and test size to be equal (20% each of overall data). # we have to define valid_size=0.5 (that is 50% of remaining data). A new class LogisticRegressionModel is introduced which internally implements all the following concepts to train and test the data. After initialization of the class, I called alog_fit method to validate the fit and predict method to trigger the classification model.

3.1.2. Sigmoid Function:

Sigmoid function is used to classify the binary dataset. It can be shown as sigmoid (z) = 1/(1/e^-z) where z is the linear classification function that needed to be applied for. Hence the value is bound between 0 and 1 such that, if the resultant value is greater than 0.5, the output is displayed as 1 else 0.

Code Implementation

```
# Basic sigmoid function.
def sigmoid_function(self, x):
    return 1/(1 + np.exp(-x))
```

3.1.3. Cost Function:

Cost function is basically the error displayed by the model with respect to the predicted value. To achieved good model, the cost function error rate must be reduced to lowest i.e., attaining global minima.

```
# Cost funciton to calculate
def cost_func(self, A, Y):
    cost = (-1/Y.shape[1])*(np.dot(Y, np.log(A).T) + np.dot(1-Y, np.log(1-A).T))
    cost = np.squeeze(cost)
    return cost
```

3.1.4. Forward and Back Propagation:

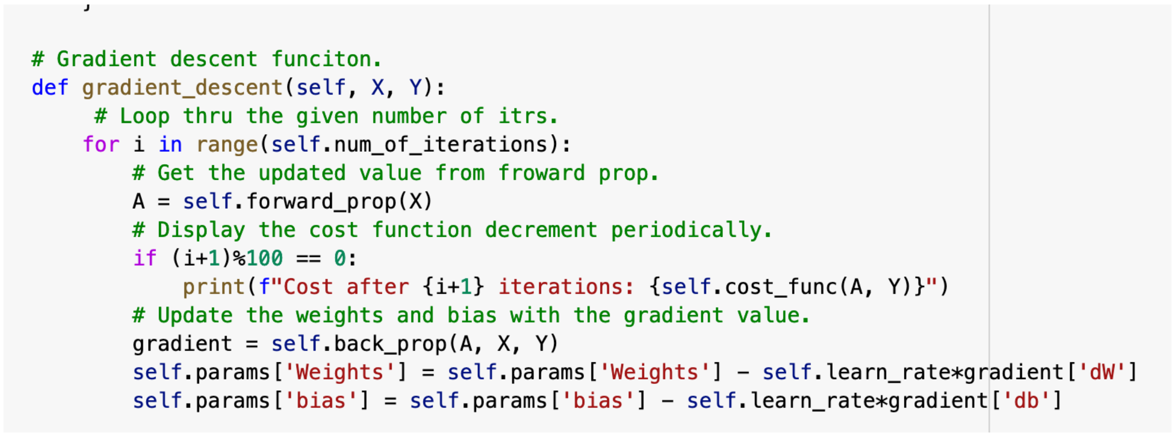
The weights and bias are to be updated to reduce the error rate. Hence, we need to use forward and backward propagation in the model to

```python
# Perfrom forward propagation and retrun new sigmoid function val.
def forward_prop(self, X):
    Weights = self.params['Weights']
    bias = self.params['bias']
    return self.sigmoid_function(np.dot(Weights, X) + bias)

# Back propagation
def back_prop(self, A, X, Y):
    m = Y.shape[1]
    dW = np.dot(A - Y, X.T)/m
    db = np.sum(A - Y, axis=1, keepdims=True)
    # Return the results
    return {
        "dZ": A - Y,
        "dW": dW,
        "db": db
    }
```

3.1.5. Gradient Descent

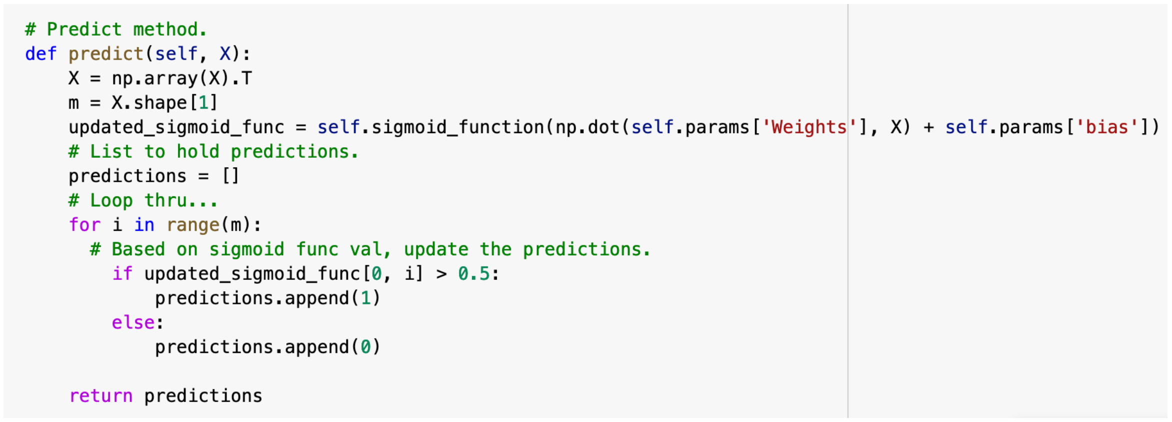
Training the model such that the weights and bias must be achieved to optimal value where the error rate is low and close to global minima. To achieve this, the formula for calculation of updated weights is $W = (Y_{pred} - Y)*X$. Where Y_{pred} is the predicted output and X is the given features in the input matrix. Which can be calculated by applying sigmoid to the given function. And similarly, Bias can be calculated by $B = (Y_{pred} - Y)$.



```python
# Gradient descent funciton.
def gradient_descent(self, X, Y):
     # Loop thru the given number of itrs.
    for i in range(self.num_of_iterations):
        # Get the updated value from froward prop.
        A = self.forward_prop(X)
        # Display the cost function decrement periodically.
        if (i+1)%100 == 0:
            print(f"Cost after {i+1} iterations: {self.cost_func(A, Y)}")
        # Update the weights and bias with the gradient value.
        gradient = self.back_prop(A, X, Y)
        self.params['Weights'] = self.params['Weights'] - self.learn_rate*gradient['dW']
        self.params['bias'] = self.params['bias'] - self.learn_rate*gradient['db']
```

3.1.6. Prediction:

I then called the model to predict the functions



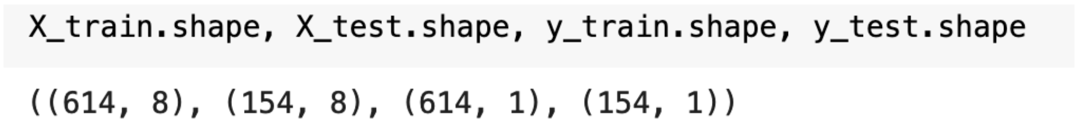
```python
# Predict method.
def predict(self, X):
    X = np.array(X).T
    m = X.shape[1]
    updated_sigmoid_func = self.sigmoid_function(np.dot(self.params['Weights'], X) + self.params['bias'])
    # List to hold predictions.
    predictions = []
    # Loop thru...
    for i in range(m):
      # Based on sigmoid func val, update the predictions.
        if updated_sigmoid_func[0, i] > 0.5:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

3.1.7. Visualization:

The following intermediatory outputs shows while training and testing the regressor classifier.

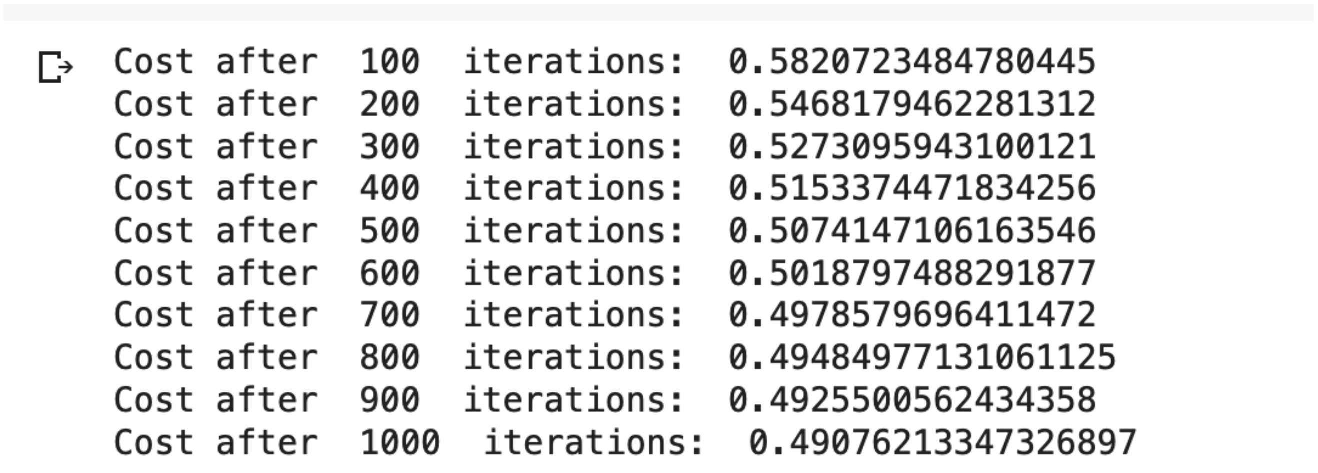
Splitting data set for test and train



```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

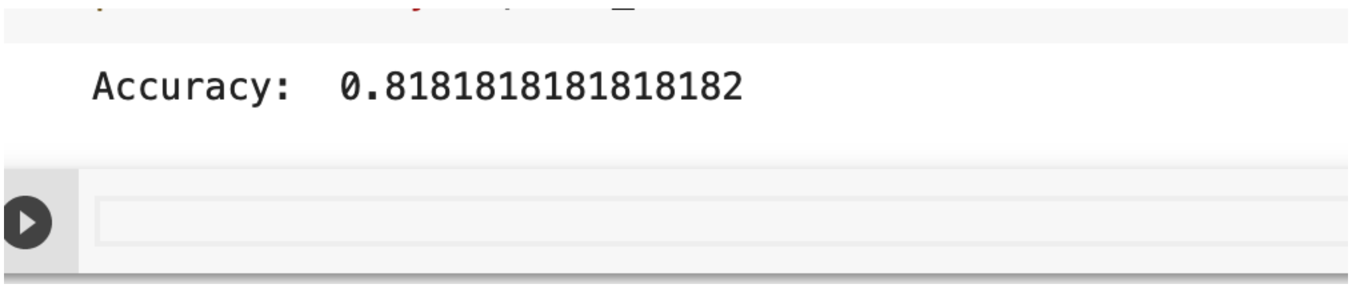
```
((614, 8), (154, 8), (614, 1), (154, 1))
```

Gradient descent. Reducing the error percent.



```
Cost after  100  iterations:  0.5820723484780445
Cost after  200  iterations:  0.5468179462281312
Cost after  300  iterations:  0.5273095943100121
Cost after  400  iterations:  0.5153374471834256
Cost after  500  iterations:  0.507414710616354б
Cost after  600  iterations:  0.5018797488291877
Cost after  700  iterations:  0.4978579696411472
Cost after  800  iterations:  0.49484977131061125
Cost after  900  iterations:  0.4925500562434358
Cost after  1000  iterations:  0.49076213347326897
```

Accuracy of the model.



```
Accuracy:  0.8181818181818182
```

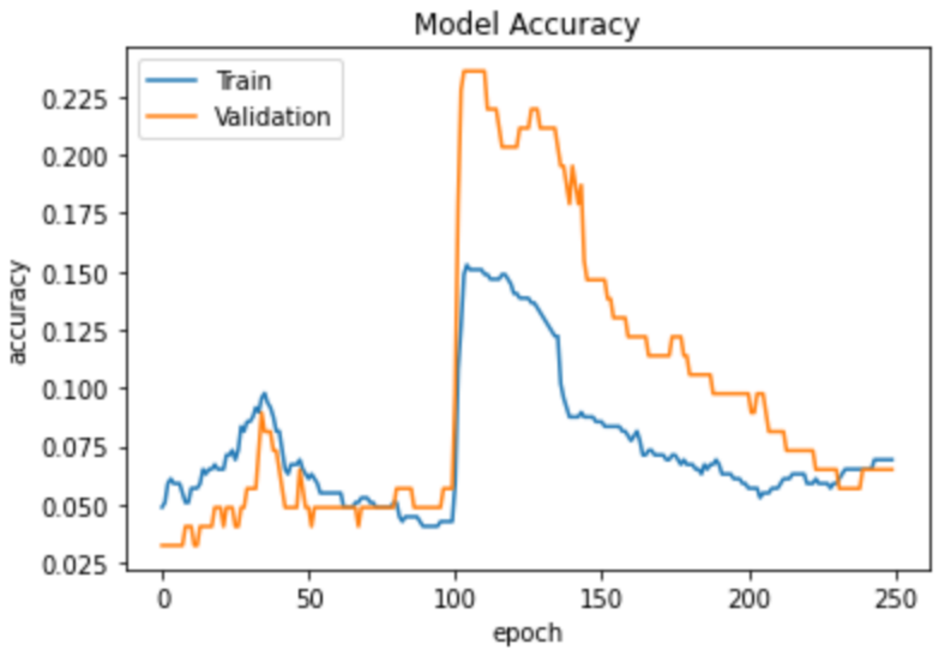
3.1.8. Conclusion:

So I tried to implement regressor_2 to our test data with learning_rate=0.01 with number of iterations of 1000, I got an accuracy of 81.81

3.2. PART 2: Implementation of Neural Networks

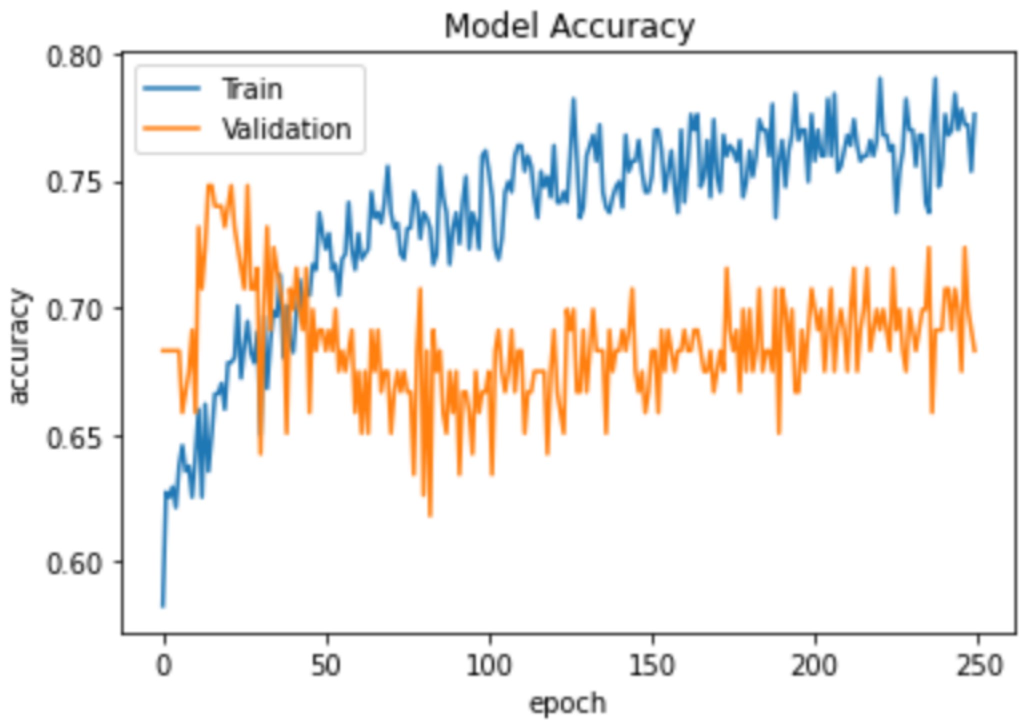
For Neural Networks upon implementing baseline mode with 2 hidden layers having 32 and 16 units, I got the accuracy as 76.54%. Adding l1 regularization increases the performance to 78.6

Below depicted is the graph for accuracy model after adding l1 regularization.



3.2.1.

3.2.2. Model accuracy with dropout



3.2.3.

3.2.4.