Logistic Regression on Pima Indians Diabetes Database dataset

Abstract

This project is an effort to use Logistic Regression, written from the ground up, to learn from the Pima Indians Diabetes Database dataset and predict if a patient has Diabetes or not based on the diagnostic measures supplied in the dataset. Gradient Descent is used to optimize weights and bias. On the test set, an accuracy of 80% is obtained.

1.Introduction

Pima Indians with type 2 diabetes are **metabolically characterized by obesity, insulin resistance**, insulin secretory dysfunction, and increased rates of endogenous glucose production, which are the clinical characteristics that define this disease across most populations. In this study, variables such as glucose levels, skin thickness, and insulin levels are used to determine whether a person has diabetes.

2.Dataset

Pima Indians Diabetes Database dataset will be used for training, and testing. The dataset contains medical data of female patients above the age of 21 and 768 instances with the diagnostic measurements of 8 features. The 8 features are as follows:

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)

3.Data Preprocessing

- The given dataset contains 768 instances, splitting this data into training, test, and validation sets.

  Testing data = 60% of Total data
  Validation data = 20% of Total data
  Test data = 20% of Total data

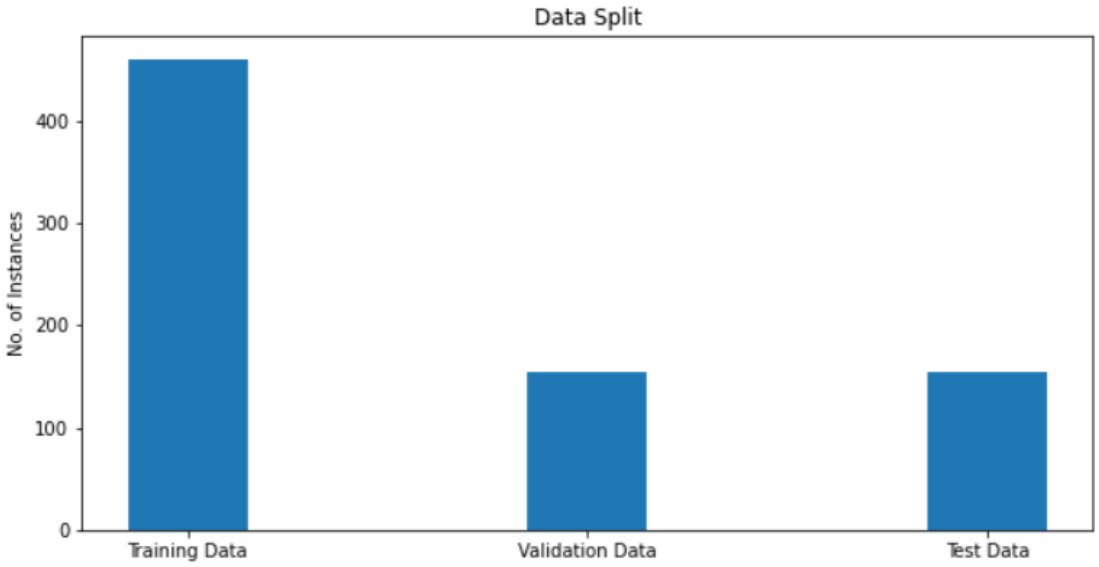


Figure 1

3.1 Imputation and Correlation

- There are many instances with zero values for some features, which is practically not possible.

Feature	No. of instances with 0 as value
Pregnancies	111
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0

- To avoid the data leakage and improve model performance these zero values are imputed with mean of the respective column.
- The correlation between these features improves after imputation, represented in Figure2.

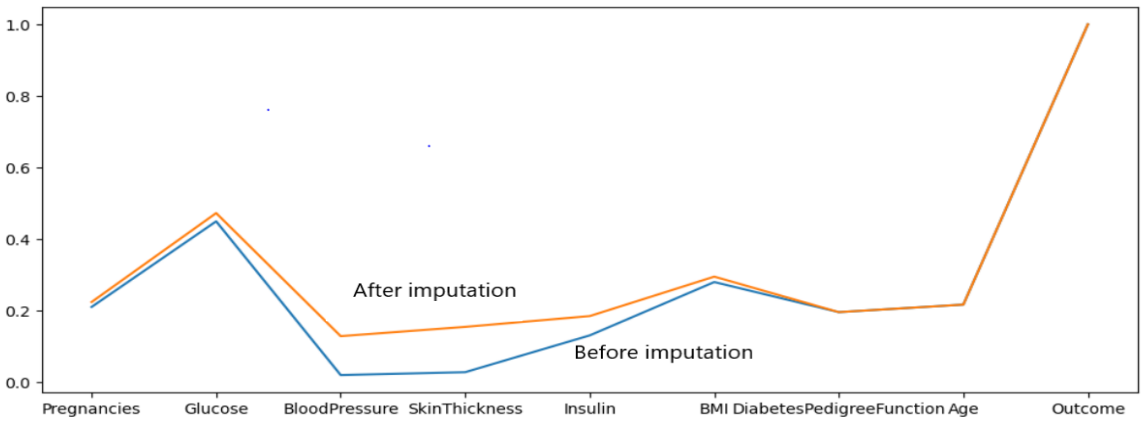


Figure 2

3.2 Normalization

- The values of each feature are in different ranges

```
In [4]: diabetes_data.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Figure 3

```
(diabetes_data[:]==0).sum()
```

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
```

From the above data it is observed that each feature has different ranges, so normalizing the data and bringing every value in the range of 0-1 would help the model to process the data accurately.

$$y = (x - \min(x))/(\max(x) - \min(x))$$

y: cell value of each column

Every cell value will be normalized, and the range of every column will be between 0 to1

- The normalization is done after the data split, the train data min max values are used to normalize test and validation data.

4.Model Architecture

4.1 Logistic Regression

Logistic regression is one of the most common machine learning algorithms used for binary classification. It predicts the probability of occurrence of a binary outcome using a logit function. It is a special case of linear regression as it predicts the probabilities of outcome using log function.

We use the activation function (sigmoid) to convert the outcome into categorical value.

Sigmoid Function

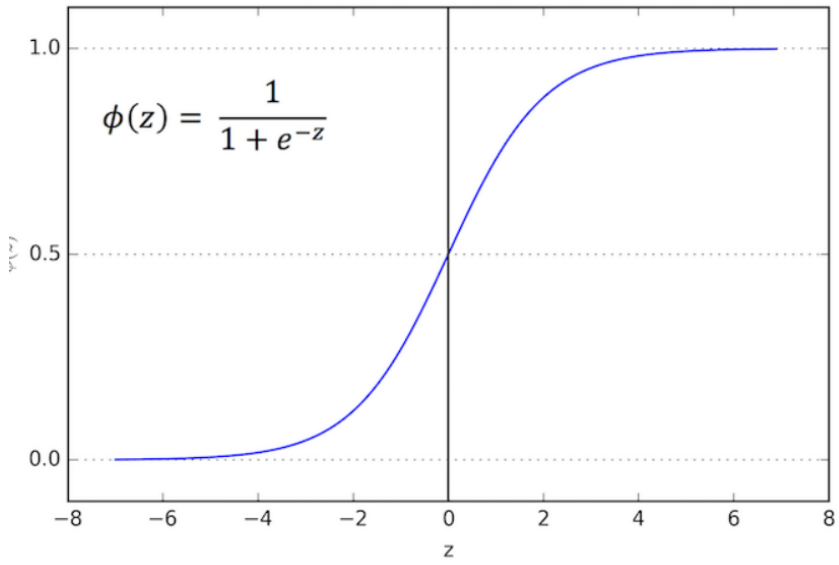


Figure 4

4.2 Cross entropy log loss

The cross-entropy loss function is an optimization function used for training machine learning classification models that classify data by predicting the likelihood (value between 0 and 1) of belonging to one class or another. When the projected probability of class differs significantly from the actual class label (0 or 1), the cross-entropy loss is large.

It is a single value representation of performance of the model, The below cost function is used in this model by gradient decent to optimize the weights and bias.

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

4.3 Gradient Descent

Gradient Descent is used in this project to optimize weights and bias. The gradient of the loss function is computed, and each weight is reduced by the product of the gradient and the learning rate. The learning rate is assumed to be 0.1.

$$\frac{\partial}{\partial w}J(w) = \nabla_w J \qquad w = w - \alpha\nabla_w J$$

$$\frac{\partial}{\partial b}J(w) = \nabla_b J \qquad b = b - \alpha\nabla_b J$$

5. Hyper Parameters Tuning

In this project batch size and the learning rate are the hyper parameters, trying to find the weights and bias for different batch size and learning rates would help us understand what the optimal hyper parameters are.
iterations = [3000,10000,15000]
learning_rates = [0.1, 0.05, 0.06]
validation and train loss are calculated with different batch sizes and learning rates, it would give 9 such graphs for the above mentioned combinations.

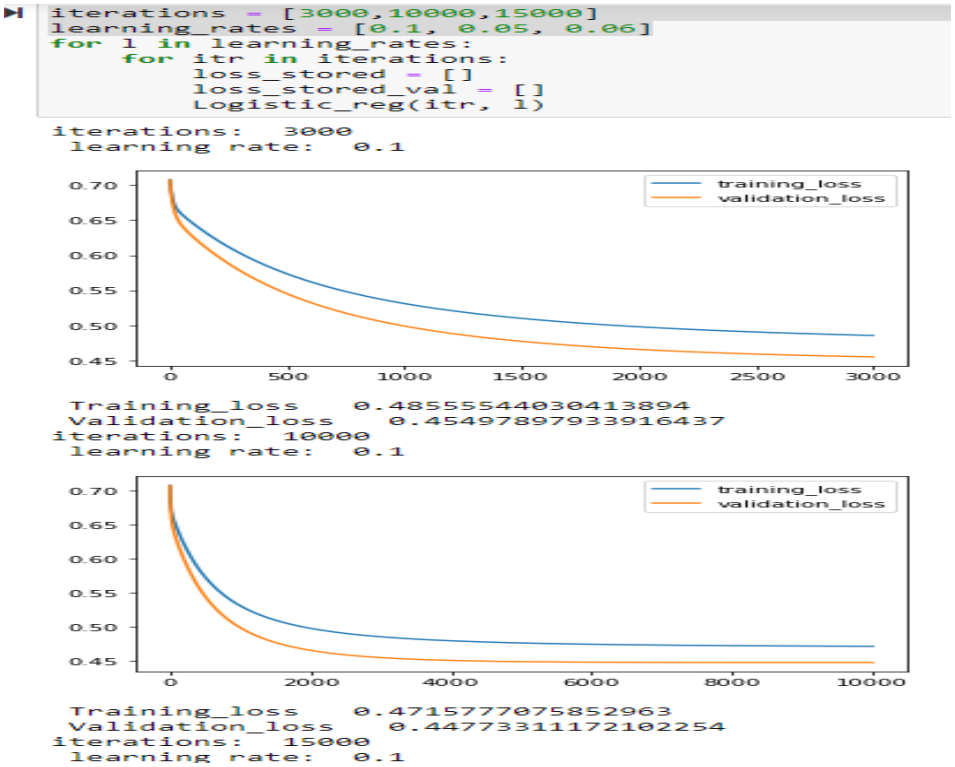


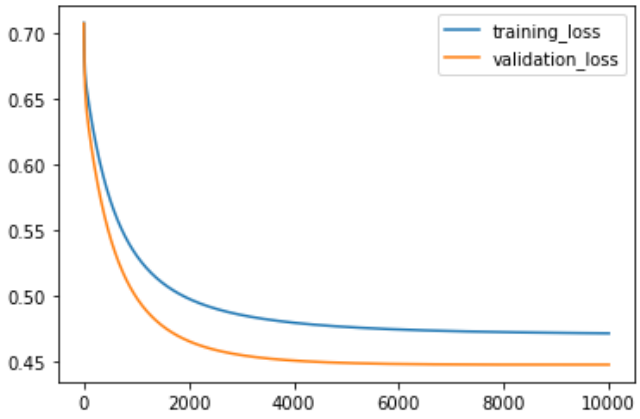
Figure 5

After examining the training and validation loss graphs for various batch sizes and learning rates, batch size =10000 and learning rate =0.1 would result in less loss.

6.Results

6.1 Training Data & Validation Data

Batch size =10000 and learning rate =0.1. Initial weights are a zero matrix, and the bias is 0.1




```
Plot to represent Training and Validation loss
 Training_loss    0.4715777075852963
 Validation_loss    0.44773311172102254
```

Figure 6

6.2 Testing

Batch size =10000 and learning rate =0.1. Initial weights are a zero matrix, and the bias is 0.1
The model gave an accuracy of 80% over the test data, considering any probable value over 0.5 as 1.

Calculating Training, validation accuracy.

```
#Train data accuracy
y_predicted=predict(train_data_norm_X,weights,cnst)
train_accuracy=predicted_accuracy(train_data_norm_y,y_predicted)*100
print("Taining data accuracy: ",train_accuracy)

#Validation data accuracy
y_predicted=predict(validation_data_norm_X,weights,cnst)
validation_accuracy=predicted_accuracy(validation_data_norm_y,y_predicte
print("Validation data accuracy: ",validation_accuracy)
```

```
Taining data accuracy:  77.39130434782608
Validation data accuracy:  77.92207792207793
```

Testing the model with Test data

```
#Test data accuracy
y_predicted=predict(test_data_norm_X,weights,cnst)
test_accuracy=predicted_accuracy(test_data_norm_y,y_predicted)*100
print("Test data accuracy: ",test_accuracy)
```

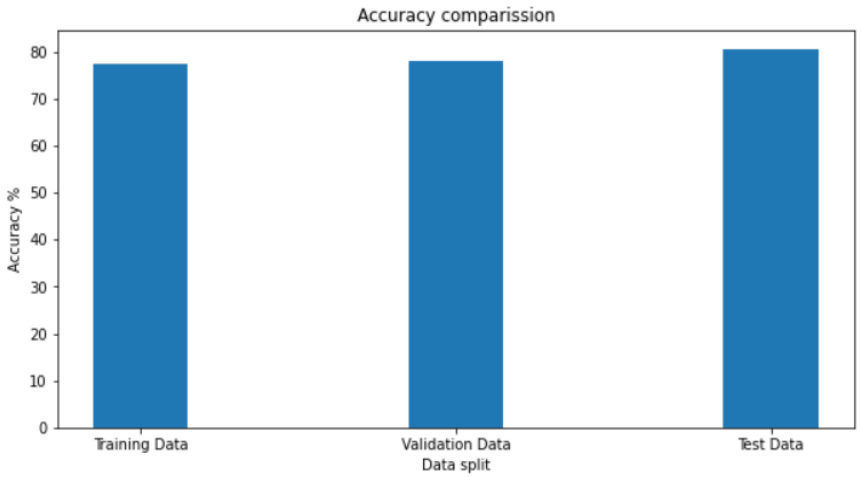
```
Test data accuracy:  80.51948051948052
```

7. Evaluation Metrics

Accuracy is measured by calculating the sum of predicted instances that matches the actual instances(outcomes) and dividing it by number of outcomes.

```
def predicted_accuracy(actual_data_y,pred_data_y):
    prob=np.sum(actual_data_y==pred_data_y)/len(actual_data_y)
    return prob
```

Conclusion



We can confidently state that the model is neither overfitting nor underfitting because the validation loss was lower than the training loss. Furthermore, on unseen data from the testing set, the model had an accuracy of 80%. As a result, if the model is used in the real world, it can be confidently stated that it will perform well.

References:

https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

https://financial-engineering.medium.com/logistic-regression-without-sklearn-107e9ea9a9b6

https://www.analyticssteps.com/blogs/introduction-logistic-regression-sigmoid-function-code-explanation

https://towardsdatascience.com/end-to-end-data-science-example-predicting-diabetes-with-logistic-regression-db9bc88b4d16

https://machinelearningmastery.com/difference-test-validation-datasets/