Implementing Linear Regression Classifier on WDBC Dataset

Gursimran Singh
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
gursimr2@buffalo.edu

Abstract

This project is intended to be considered as project 1 submission for IML CSE_574 coursework. We have to classify suspected FNA cancer cells in Wisconsin Diagnostic Breast Cancer dataset into Benign or Malignant. We are using Logistic Regression Classifier to train our model on the cancer data. Experimental results on the dataset, which comprises of 569 characteristics of the cell nuclei present in the digitized image of Fine Needle Aspirate(FNA). The model predicted class labels of unseen data with 98% accuracy.

1 Introduction

"Anything that could give rise to smarter-than-human intelligence—in the form of Artificial Intelligence, brain-computer interfaces, or neuroscience-based human intelligence enhancement - wins hands down beyond contest as doing the most to change the world. Nothing else is even in the same league." —Eliezer Yudofsky

Classification techniques are an essential part of machine learning and data mining applications. Approximately 70% of problems in Data Science are classification problems. A popular classification technique to predict binomial outcomes (y = 0 or 1) is called Logistic Regression. Logistic regression predicts categorical outcomes (binomial/multinomial values of y), whereas linear Regression is good for predicting continuous-valued outcomes (such as the weight of a person in kg, the amount of rainfall in cm).
In this project we use some Logistic Regression Features for implementing our model such as sigmoid function, which will be used to give the probability for target labels. This will be discussed in following sections.

2 Dataset

Wisconsin Diagnostic Breast Cancer (WDBC) dataset will be used for training, validation and testing. The dataset contains 569 instances with 32 attributes (ID, diagnosis (B/M), 30 real-valued input features). Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. Computed features describes the following characteristics of the cell nuclei present in the image:

1	radius (mean of distances from center to points on the perimeter)
2	texture (standard deviation of gray-scale values)
3	perimeter

4	area
5	smoothness (local variation in radius lengths)
6	compactness (perimeter2/area − 1.0)
7	concavity (severity of concave portions of the contour)
8	concave points (number of concave portions of the contour)
9	symmetry
10	fractal dimension (“coastline approximation” - 1)

The mean, standard error, and “worst” or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features.

Data Set Characteristics:	Multivariable	**Number of instances:**	569	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	32	**Date Donated:**	1995-11-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1001536

3 Preprocessing

Data Preprocessing is implemented in this project in the following manner.

1. Dropping column ‘id’ from dataframe.
2. Mapping diagnosis column values to 0 and 1.
3. Analyzing given data features by plotting graphs.
4. Visualizing correlation between features using heatmap.
5. Performing data wrangling
6. Z-Score Normalization/Standardization
7. Data Splitting

3.1 Drop Column

The feature ‘id’ has no role in determining our target feature, i.e., diagnosis, so we are dropping it using the following python script.

Drop Column id

```
cancer_data.drop(['id'],axis=1,inplace=True)
```

3.2 Mapping

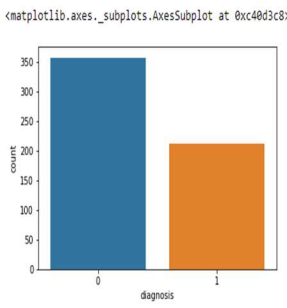
We need to map diagnosis feature value to 0 and 1, because it is a categorical data and we need discrete values to train our model.

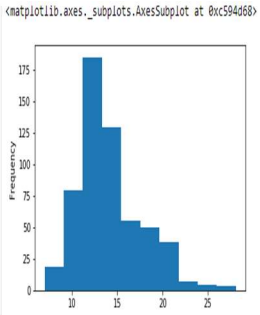
Map diagnosis column to 0 and 1

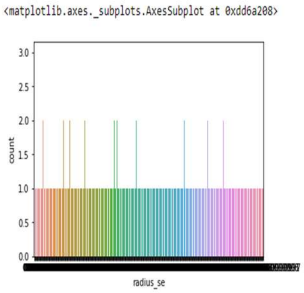
```
]: cancer_data['diagnosis']=cancer_data['diagnosis'].map({'M':1,'B':0})
```

3.3 Analyzing Data

We need to analyze the given dataset to understand it before performing any operations. The following graphs shows the initial data statistics for few features. We can use matplotlib library for this.

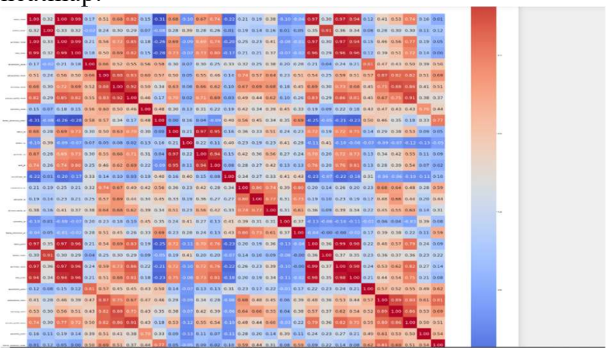






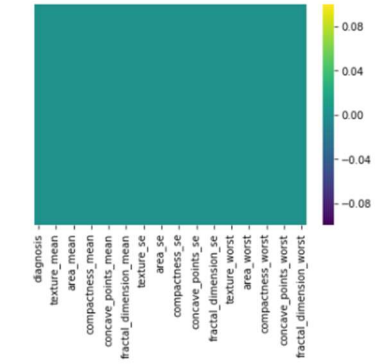
3.4 Visualizing Correlation

The following graph shows correlation in different features of the given dataset using heatmap.



3.5 Data Wrangling

In this check we check the dataset for any missing or null data. The following graph shows the if any of the features has null values.



3.6 Z-Score Normalization

A z-score is the number of **standard deviations** from the mean a data point is. Z-scores are a way to compare results from a test to a "normal" population. Z-scores range from -3 standard deviations (which would fall to the far left of the normal distribution curve) up to +3 standard deviations (which would fall to the far right of the normal distribution curve).The basic z score formula for a sample is:

z = (x – μ) / σ

It is implemented as follows:

Standardization/ Z-Score Normalization

```
]: for col in cancer_data.columns[2:32]:
       cancer_data[col] = (cancer_data[col] - cancer_data[col].mean())/cancer_data[col].std(ddof=0)
```

3.7 Splitting Data

We are given 569 record for 30 independent feature for training and testing our model. We will split the given data into 3 sets: Training, Test, and Validation.
Training set is the data we use to train our model and major portion of the given data will be contributed to this set.
Validation set ,as the name suggests, validates the results of model on training data, and signifies the accuracy of model in order to improve it and tune its hyperparameters, so that we get best results in the test set.
Test set is used for testing the model we trained on our training data. We predict the target value for test set based on features in test set, and our aim is to match it with the given target values.
In the given dataset, we split our data into 80% , 10% and 10% for training , validation and testing respectively.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1,random_state=1)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size = 0.111,random_state=1)
```

4 Architecture

The Architecture of our Linear Regression model comprises of the following main features:

1. Hypothesis Function
2. Log Likelihood Loss
3. Sigmoid Function
4. Gradient Descent

4.1 Hypothesis Function

The **linear equation** is the standard form that represents a straight line on a graph, where m represents the **gradient- how steep the line is**, and b represents the **y-intercept- where the line crosses the y-axis**.

$$y = mx + c$$

The Hypothesis Function is the exact same function in the notation of Linear Regression.

$$h_\theta(x) = \theta_1 x + \theta_0$$

The two variables we can change — m and b — are represented as parameters θ_1 and θ_0.

In the beginning, we **randomly initialize our parameters**, which means we give θ_1 and θ_0 random values to begin with. The python code is as follows

```
z = np.dot(w.T, X) + b
```

4.2 Log Likelihood Loss

To calculate, how good or bad our model is, we need to implement a loss function on our training data, which will give us a numerical loss value to justify the goodness of our model. On the basis of loss function we improve our hyper-parameters to get minimum loss function. For this model, we have used this loss function:

$$L(p,y) = -(ylogp + (1-y)\,log(1-p))$$

Loss function is the loss for just for training example. Cost function is the loss for whole training set.

```
cost = -np.sum(np.multiply(np.log(p), Y) + np.multiply((1 - Y), np.log(1 - p)))/float(m)
```

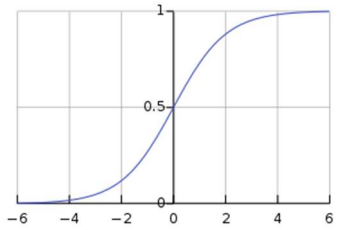
4.3 Sigmoid

We use this equation to predict our values for each sample of data:

$$y(x,\mathrm{w}) = \sum_{j=0}^{M-1} w_j \phi_j(x) = \mathrm{w}^T \phi(x)$$

But this equation gives the output in the range (-∞,+∞). So to reduce the output range to (0,1), we use sigmoid function:

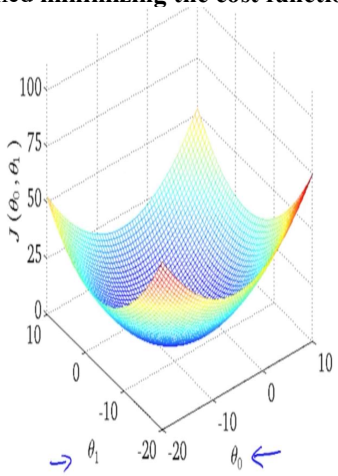
$$f(t) = \frac{1}{1+e^{-t}}$$




```
p = sigmoid(z)
```

4.4 Gradient Descent

Our goal with linear regression was to find the line which best fits a set of data points. In other words, it's the line that's the least incorrect or has the lowest error. If we graph our parameters against the error (i.e graphing the cost function), we'll find that it forms something similar to the graph below. At the lowest point of that graph, the error is at it's lowest. Finding this point is called **minimizing the cost function**.



To do this, we need to consider what happens at the bottom of the graph — the gradient is zero. So **to minimize the cost function, we need to get the gradient to zero.**The gradient is given by the derivative of the function, and the partial derivatives of the functions are:

$$\frac{\delta}{\delta\theta_0} = h_\theta(x_i) - y_i$$

$$\frac{\delta}{\delta\theta_0} = (h_\theta(x_i) - y_i)x_i$$

Python code:

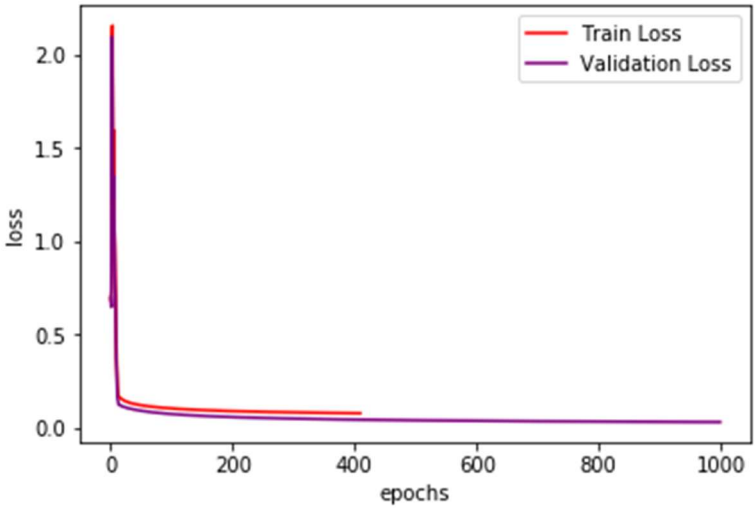
```
dw = (1 / float(m)) * np.dot(X, dz.T)
db = (1 / float(m)) * np.sum(dz)
```

5 Results

The very first results are shown in the following graph when the hyper-parameters are:

epochs = 1000

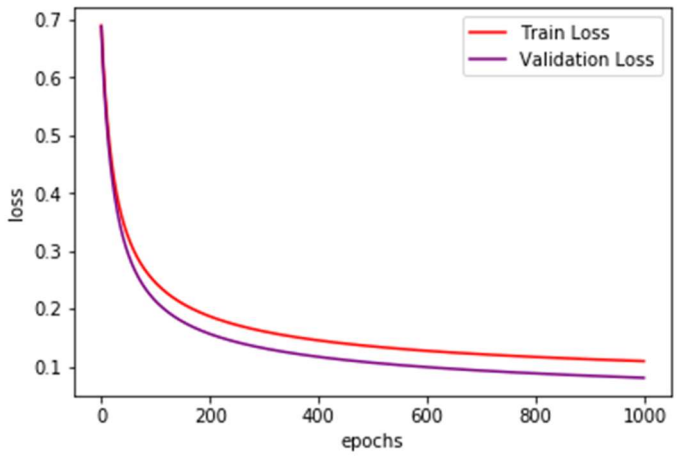
learningrate = 0.1



The final results are shown in the following graph when the hyper-parameters are:

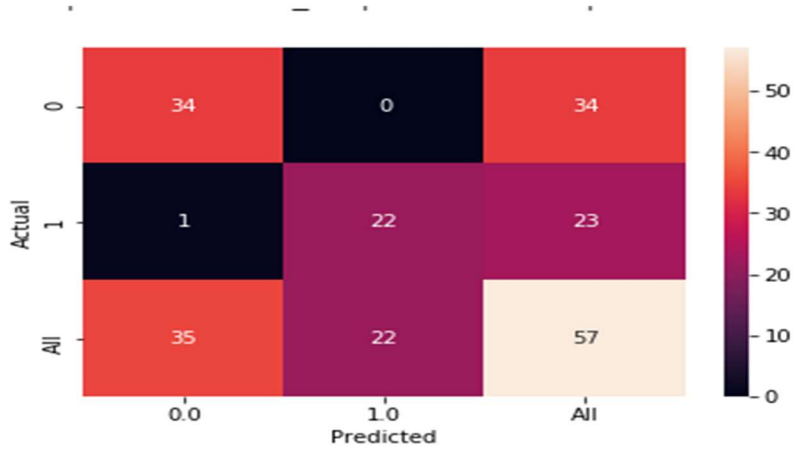
epochs = 1000

learningrate = 0.01



The prediction is based on that the outcome values of sigmoid function on test data and updated theta and bias, which are **rounded to 1 if greater than 0.5 and to 0 if less than 0.5**.

The confusion matrix and evaluation metrics of our model predictions given the test data is



Performance metrics for Classification

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matri
```

```
a=accuracy_score(y, predict)
print "Accuracy is",a
```

```
Accuracy is 0.9824561403508771
```

```
p=precision_score(y, predict)
print "Precision is",p
```

```
Precision is 1.0
```

```
r=recall_score(y, predict)
print "Recall is",r
```

```
Recall is 0.9565217391304348
```

```
f=f1_score(y, predict)
print "F-measure is",f
```

```
F-measure is 0.9777777777777777
```

6 Conclusion

The given model has validation loss less than the training loss which shows that it is neither overfitted nor underfitted and with an accuracy of 98%, is ready to be implemented in real world applications.