Logistic Regression for Cancer Dataset

Sachin Venugopal
University at Buffalo
sachinve@buffalo.edu

Abstract

This project is an implementation of Logistic Regression to solve a two-class classification problem, where we train a model to learn from a cancer dataset and predict whether a given tumor is Malignant or Benign. The dataset used here is the Wisconsin Diagnostic Breast Cancer (wdbc.dataset) where the features used for classification are pre-computed from images of a fine-needle aspirate (FNA) of breast mass.

1 Introduction

Given a dataset of multiple features, the task of classifying each sample into one of two classes can be performed by using Logistic Regression. Logistic Regression is a statistical method, commonly used as a Machine Learning algorithm, to solve binary classification problems. It derives its name from the use of Logistic Sigmoid or 'Logit' function given by,

$$\boldsymbol{\sigma(z) = \frac{1}{1 + e^{-z}}}$$

In Logistic Regression, we use the sigmoid function to describe the probability that a sample belongs to one of two classes [1]. The sigmoid function returns a value ranging from 0 to 1. We can set a decision boundary to determine which class the output of the sigmoid function belongs to. Here, we use a decision boundary of 0.5, that is if the value of the sigmoid function on our input is ≥ 0.5, we assign a value of 1 to the result (tumor is Malignant). Else, we assign a value of 0 (tumor is Benign).

2 Dataset

We use Wisconsin Diagnostic Breast Cancer (WDBC) dataset for the purpose of training, validation and testing our model. The dataset consists of 569 instances, each have 10 key features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass, represented in terms of mean, standard error and "worst" or largest mean of the three largest values (hence 30 real-valued features). The dataset also contains a record of the diagnosis - whether the tumor is Malignant (M) or Benign(B) for each instance and the ID for each instance. We process the dataset extracting the features and diagnosis from it in order to train, test and validate our model.

3 Pre-Processing

The dataset provided contains 30 key features which we would like to use for training, validation and testing of our model. The dataset is read into a Pandas Dataframe. We preprocess the dataset in 3 steps:

1) **Removing unnecessary data**: We drop the 'ID' column from the dataset since it is inconsequential to our prediction.
2) **Partitioning the dataset**: We now partition the dataset in two stages:
    a. The Diagnosis column in the dataset contains values our model needs to predict. A value of 'M' indicates that the tumor is malignant and a value of 'B' indicates that the tumor is benign, respectively, for each of the samples in the dataset. The remaining 30 columns in the dataset are the 30 feature vectors we will use as input to our model to predict the output. We therefore partition the dataset into two parts:
        i. Y_data: An M*1 matrix (where M is the total number of samples) which contains the result for each sample
        ii. X_data: An M*N Matrix (where M is the total number of samples and N is the number of features) of 30 feature sets.
    b. The data is now randomized and divided into 3 parts, for the purpose of training, validating and testing the model. We randomize the data in order to eliminate any ordering provided in the dataset so that the model can be trained and validated over a random distribution of sample data. We use the Sci-kit Learn library function train-test-split() in order to split the data into 3 parts of 80% (training data), 10% (validation data) and 10%(testing data). We now have the following data:
        - Y_train and X_train Matrices: For training
        - Y_validate and X_validate Matrices: For validation
        - Y_test and X_test: For testing

3) **Normalizing the data**:
    a. The data in the Response("Y") Matrices contain values of M(Malignant) and B(Benign). We translate it to 1 and 0, respectively, for ease of calculation.
    b. The data in the Feature("X") Matrices range from columns of integers in the tenth and hundredth digit to decimal values. We need to normalize the data for ease of calculation and representation. We use the preprocessing.MinMaxScaler() function provided by the Sci-kit Learn library for this purpose.

4 Architecture

4.1. Sigmoid Function

We use Logistic Regression as the classification algorithm to assign our observations to two distinct classes. Since our input data is continuous, we use linear regression along with an activation function used to classify the predicted outputs to either 0 or 1. The activation function used here is the Sigmoid function.

For our continuous inputs, assume class-conditional densities with some covariance matrix $\sum$ given by,

$$p(x \mid C_k) = \frac{1}{(2\pi)^{\frac{D}{2}}} \frac{1}{|\sum|^{\frac{1}{2}}} e^{\{-\frac{1}{2}(x-\mu_k)^T \sum^{-1}(x-\mu_k)\}}$$

Consider one of the two classes for our distribution, C_1,

Using Bayes' Theorem, it's posterior probability can be written as,

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{P(x|C_1)P(C_1) + p(x|C_2)p(C_2)}$$

Dividing both numerator and denominator by $P(x|C_1)P(C_1)$,

$$p(C_1|x) = \frac{1}{1 + \frac{p(x|C_2)p(C_2)}{p(x|C_1)p(C_1)}}$$

$$= \frac{1}{1 + e^{-z}} = \sigma(z)$$

Where we have defined

$$\mathrm{z} = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$$

Here, $\sigma(z)$ is the Logistic Sigmoid function defined by,

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The sigmoid function, also called logistic function, gives an 'S' shaped curve that can take any real-valued number and map it into a value between 0 and 1. If the curve goes to positive infinity, y predicted will become 1, and if the curve goes to negative infinity, y predicted will become 0.[2]

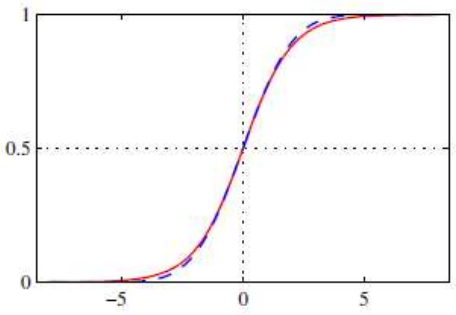


Fig 1. Logistic Sigmoid Function

We therefore define the hypothesis function as,

$$Z = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)} = W^T x + b$$

Where W^T is the weight matrix and b is the bias.

4.2. Error Function

For a data set $\{\phi_n, t_n\}$, where $t_n \in \{0, 1\}$ and $\phi_n = \phi(x_n)$, with n = 1, … … . N, the likelihood function can be written as,

$$p(t|w) = \prod_{n=1}^{N} y^{t_n}{}_n \{1 - y_n\}^{1-t_n}$$

where t = $(t_1, \dots \dots t_N)^T$ and $y_n = p(C_1|\phi_n)$

We can define an error function taking the negative logarithm of the likelihood, which gives us the cross – entropy error function, Cross-entropy error function measures the performance of a classification model whose output is a probability value between 0 and 1. It is given by,

$$E(w) = -\ln p(t|w) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

where $y_n = \sigma(a_n)$ and $a_n = w^T \phi_n$

Taking gradient of the error function with respect to w, we get,

$$\nabla E(w) = -\sum_{n=1}^{N} (t_n - y_n)\, \phi_n$$

The weight update rule is given by,
W:= $W + \alpha \nabla E(w)$

$$W := W - \alpha \nabla \sum_{n=1}^{N} (t_n - y_n)\, \phi_n$$

Here, α is the learning rate of the model.

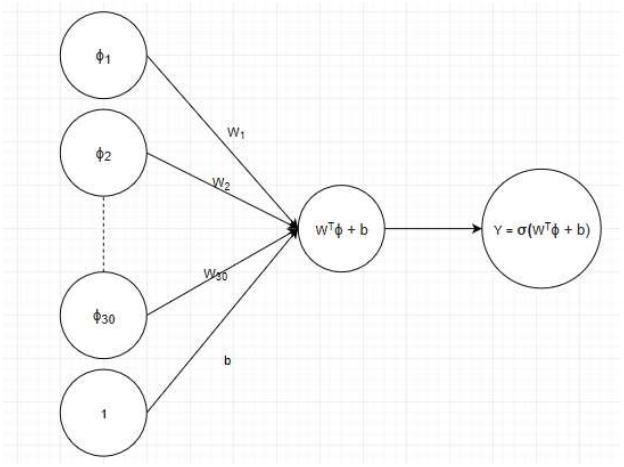


Fig2. Computational Graph

4.3. Model Training

In the implementation of the project, we train the model using the data obtained after partitioning and normalizing in the preprocessing phase. We use the hypothesis function of the form,

$$Z = W^T X + b,$$

Where, X is the feature matrix,

W^T is a weight matrix of arbitrary values,

b is a bias of arbitrary value,

Z is the response matrix

We find the prediction p for each iteration using the formula,

$$p = \sigma(Z)$$

We then use the prediction p along with the corresponding expected value in training data Y for each iteration to calculate the cost function given by,

$$\text{Cost} = -\frac{1}{m}\sum y \log p + (1-y)\log(1-p)$$

Where m is the number of samples. We divide by the number of samples in order to take an average over m values.

The value of cost determined at each iteration is then used to tune the weights using gradient descent as,

$$dz = p - y$$

$$dw = X.dz^T / m$$

$$db = \sum dz / m$$

We find the tuned values for parameters w(weights) and b(bias) using,

$$w = w - \alpha dw$$

$$b = b - \alpha db$$

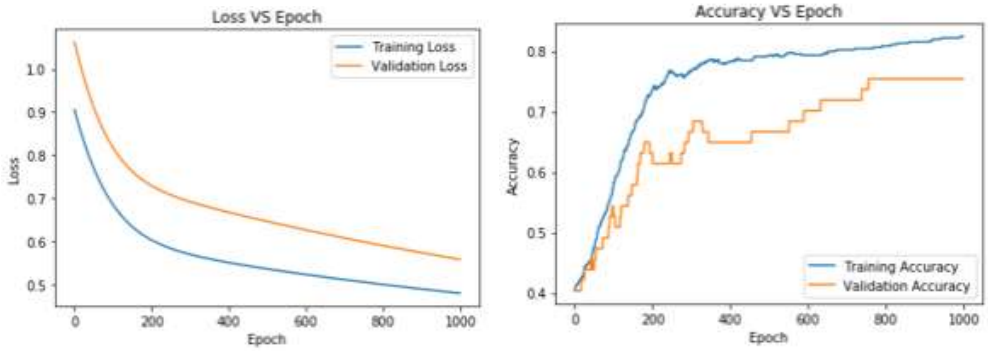
4.4. Validation and Hyperparameter Tuning

While training the model, we estimate an error value by finding the difference between the predicted value and the original response, called training error. The training error, however, gives us an estimate of how our model is performing over the training dataset only. In order to make sure that our model does not overfit the training dataset and to understand how well the model generalizes over unseen data, we use Cross-Validation. In our implementation, with each iteration of training over a set epoch value and learning rate, we use the predicted weights and bias values to test the prediction of our model with validation data. Based on the values of cost produced in both cases, we tune our hyperparameters, epoch (number of iterations) and α (learning rate), in order to obtain an optimal model.

5 Results

Using the values of cost and accuracy we obtained for different combinations of epoch and learning rate, we print Cost vs Epoch and Accuracy vs Epoch in order to estimate optimal hyperparameter values for out model.

a) For an epoch of 1000 and learning rate of 0.01, we get



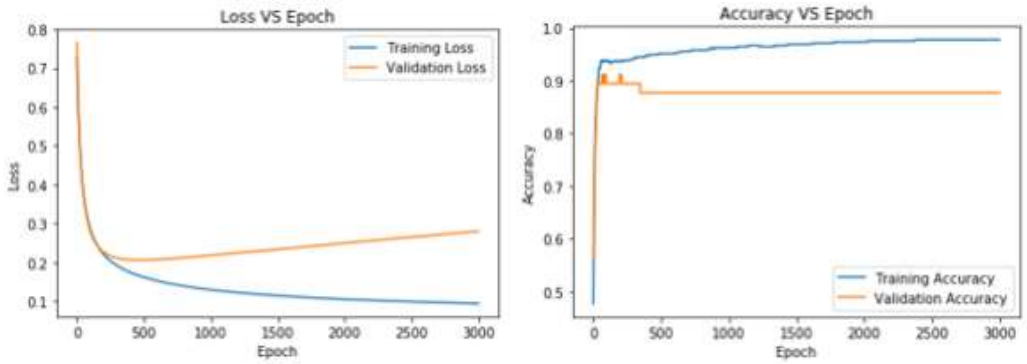
3.(a) Loss Vs Epoch and Accuracy Vs Epoch graphs

For the above combination of hyper parameter values, the minimum cost is given as:

Training Data	Validation Data
0.4802	0.5587

We observe that though the model reduces the cost overall for both the Training and Validation data, the minimum cost value can be reduced further by tuning the hyper parameter.

b) For an epoch of 3000 and a learning rate of 0.5



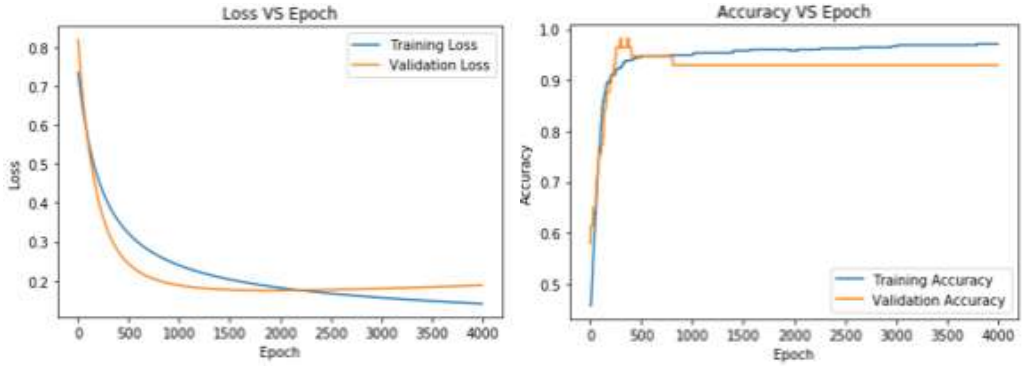
3.(b) Loss Vs Epoch and Accuracy Vs Epoch graphs

For the above combination of hyper parameter values, the minimum cost is given as:

Training Data	Validation Data
0.0938	0.2798

We observe that the model performs well in reducing the loss for Training data but does not perform as well on validation data.

c) For an epoch of 4000 and a learning rate of 0.07



3.(c) Loss Vs Epoch and Accuracy Vs Epoch graphs

For the above combination of hyper parameter values, the minimum cost is given as:

Training Data	Validation Data
0.1408	0.1880

We observe that the model performs well for both testing and validation data. We now use these hyper parameter values to test the model using the testing data,

Epoch = 4000, $\alpha = 0.07$

The performance of the model is estimated using the testing data by substituting the values of weight matrix W and bias b in the equation,

$$Y_{test} = W^T X_{test} + b$$

In order to evaluate the performance of our model, we use Accuracy, Precision and Recall given by,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Where TP = True positive, TN = True Negative, FP = False Positive, FN = False Negative

Accuracy	Precision	Recall
0.9473	1.0	0.88

7 Conclusion

We use Logistic Regression in order to train, validate and test a model to learn from WDBC dataset containing 30 features. Logistic regression uses the sigmoid function in order to classify the prediction produced by the hypothesis function as Malignant (M) or Benign(B). The model was trained, validated and tested by partitioning the randomized dataset into 80%, 10% and 10% parts. In order to obtain an optimal performance of the model, we change the values of hyper parameters, epoch and learning rate, and choose values that give us optimal performance. We then run the model on training data and evaluate the performance. As an improvement, we could use a more efficient way of varying the learning rate by introducing a decay factor, rather than doing it manually by trial and error. The decay factor ensures that the value of learning rate constantly decrements during training and we can then use the data to identify the optimal learning rate.

References

[1] https://towardsdatascience.com/logistic-regression-python-7c451928efee.

[2] https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python.

[3] 4.3.2 Logistic Regression slides, Sargur N. Srihari, University at Buffalo, State University of New York

[4] Logistic Regression and Machine Learning book, Christopher M. Bishop

[5] http://cs229.stanford.edu/notes/