CSE-574: MACHINE LEARNING PROJECT 1

BREAST CANCER PREDICTION USING LOGISTIC REGRESSION

Madhumitha Sivasankaran
(50310290)
MS in Computer Science
University at Buffalo, NY

INTRODUCTION:

Prediction of breast cancer based upon several features computed for each subject is a binary classification problem. Several discriminant methods exist for this problem, some of the commonly used methods are: Decision Trees, Random Forest, Neural Network, Support Vector Machine (SVM), and Logistic Regression (LR). Except for Logistic Regression, the other listed methods are predictive in nature; LR yields an explanatory model that can also be used for prediction, and for this reason it is commonly used in many disciplines including clinical research.

ABSTRACT:

Like most machine learning algorithms, logistic regression creates a boundary edge between binary labels. The purpose of a training process is to place this edge in such a way that most of the labels are divided so as to maximize the accuracy of predictions. The training process requires correct model architecture and fine-tuned hyperparameters, whereas data play the most significant role in determining the prediction accuracy.

In a nutshell, the idea behind the process of training logistic regression is to maximize the likelihood of the hypothesis that the data are split by sigmoid. To do this, we should find optimal coefficients for the sigmoid function. These coefficients are iteratively approximated with minimizing the loss function of logistic regression using gradient descent. As soon as losses reach the minimum, or come very close, we can use our model for prediction.

In this project, we demonstrate LR to predict breast cancer using the provided Wisconsin Diagnosis Breast Cancer (WDBC) data set.

DATASET:

The Wisconsin Diagnostic Breast Cancer (WDBC) dataset was used for training, validation and testing. The dataset contains 569 instances with 32 attributes (ID, diagnosis (Benign/Malignant) and 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.

PREPROCESSING:

The provided dataset has to undergo some initial preprocessing before the implementation of the logistic regression algorithm. Firstly, it is converted into a Pandas Dataframe which is a two dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Below (Figure 1) is the image of 5 rows and 32 columns after being converted to a dataframe.

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	25	26	27	28	29	30	
0	42302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.118
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.089
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.087
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.173
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.076

Figure 1

Secondly, the diagnosis column which has the values B and M meaning the cancer is Benign or Malignant should be converted to binary notation: ‘0’ and ‘1’ respectively (Figure 2).

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	25	26	27	28	29	30	31
0	42302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

Figure 2

The ID column, which is column 0 in the above image has to be discarded since it is not used during the computation. Also column 1 which is the diagnosis – saying whether the cancer in Benign (B) or Malignant (M) should be scraped and stored in the output vector. This can be done using iloc, which is one of the indexing methods in pandas python. This results in input feature vector ‘X’ (Figure 3) and output target vector ‘Y’. (Figure 4)

	2	3	4	5	6	7	8	9	10	11	...
0	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100	0.2419	0.07871	...
1	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170	0.1812	0.05667	...
2	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900	0.2069	0.05999	...
3	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200	0.2597	0.09744	...
4	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.104300	0.1809	0.05883	...
5	12.450	15.70	82.57	477.1	0.12780	0.17000	0.157800	0.080890	0.2087	0.07613	...
6	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.112700	0.074000	0.1794	0.05742	...

Figure 3

	1
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Figure 4

The range of data values in the dataset vary numerically. Some input values are too small or too large, we later get the logarithm of zero which equals to the negative infinitely large number and cannot be used for further computations. Thus the data values have to be normalized. This is done using the minmaxscaler() function from the sci-kit learn python package. (Figure 5)

```
[0.0226581  0.54598853 0.36373277 ... 0.91202749 0.59846245 0.41886396]
[0.27257355 0.61578329 0.50159067 ... 0.63917526 0.23358959 0.22287813]
[0.3902604  0.59574321 0.44941676 ... 0.83505155 0.40370589 0.21343303]
...
[0.62123774 0.44578813 0.30311771 ... 0.48728522 0.12872068 0.1519087 ]
[0.66351031 0.66553797 0.4757158  ... 0.91065292 0.49714173 0.45231536]
[0.50152181 0.02853984 0.01590668 ... 0.         0.25744136 0.10068215]
```

Figure 5

Now, the data has to be split into training, test and validation sets. In order to perform this operation, we use the train_test_split function from the sci-kit learn python package. The size of the test and validation samples should be 20% of the whole data. We use this sample to validate training results and to be sure that there is no overfitting. Also, we shuffle the data we are splitting, to be sure that training and test data are of the same distribution.

ARCHITECTURE:

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

In order to map predicted values to probabilities, we use the sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

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

S(z): output between 0 and 1 (probability estimate)

Z = input to the function (your algorithm's prediction ex: mx+b)

e = base of natural log

Our current prediction function returns a probability score between 0 and 1. In order to map this to a discrete class (true/false), we select a threshold value or tipping point above which we will classify values into class 1 and below which we classify values into class 2.

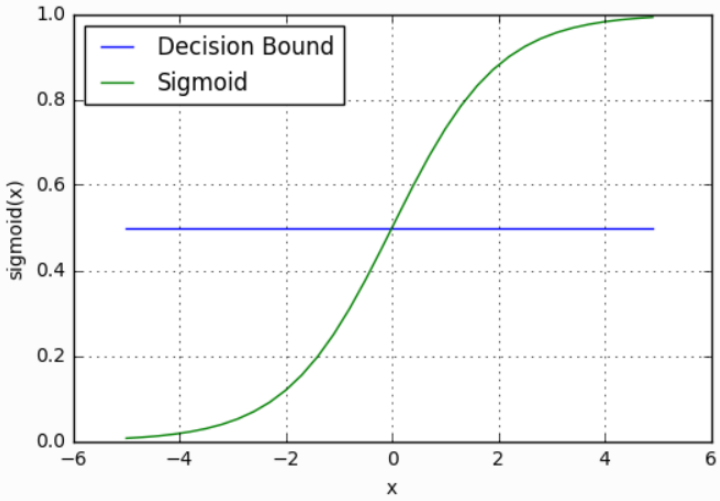

Figure 6

From sigmoid functions and decision boundaries, we can now write a prediction function. A prediction function in logistic regression returns the probability of our observation being positive, True, or "Yes". We call this class 1 and its notation is P(class=1). As the probability gets closer to 1, our model is more confident that the observation is in class 1.

We need to define the cost function for the logistic regression. The most common approach is to iterate over training examples to apply sigmoid to them, then iterate one more time to count the sum of losses.

However, we use numpy to apply sigmoid to the whole array and count losses of all the array. Cross-entropy loss can be divided into two separate cost functions: one for y=1 and one for y=0.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$

$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$$

The idea of cost function is that we count the sum of the metric distances between our hypothesis and real labels on the training data. The more optimized our parameters are, the less is the distance between the true value and hypothesis.

The next step is called a stochastic gradient descent. This is the main part of the training process because at this step we update model weights. Here we use the hyperparameter called learning rate, that sets the intensity of the training process. Learning rate is very important hyperparameter and has to be selected very carefully. Using too small learning rate, you can move too slow and never reach the minimum of the loss function, while using too large a learning rate may cause you to miss the minimum entirely.

If our model is working, we should see our cost decrease after every iteration.

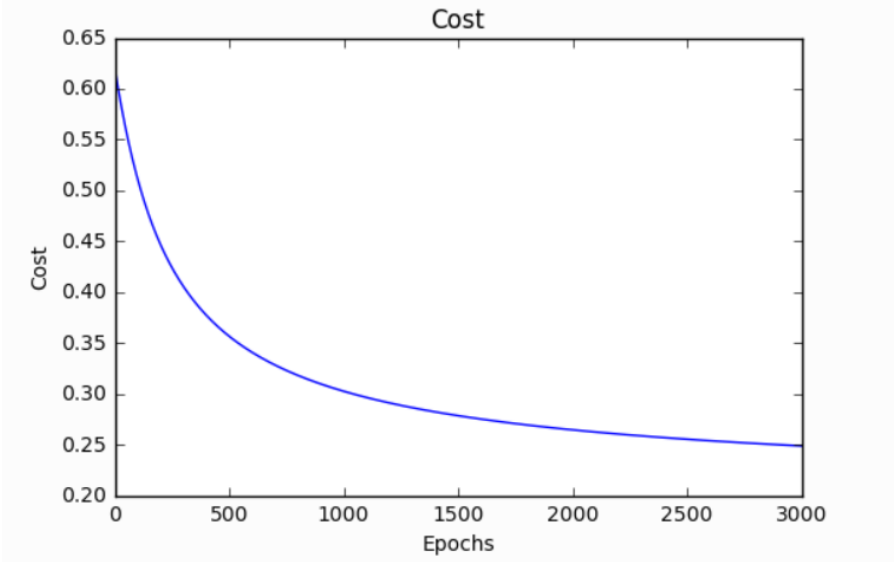

Figure 7 : Sample plot between Cost and Epochs

RESULT AND OBSERVATIONS:

Based on the given dataset, the samples have been trained, tested and validated. The 80 percent of data that was used for training resulted an accuracy of `98.46153846153847 %`.

Below is the plot of Train Accuracy Vs Epochs (Figure 8)

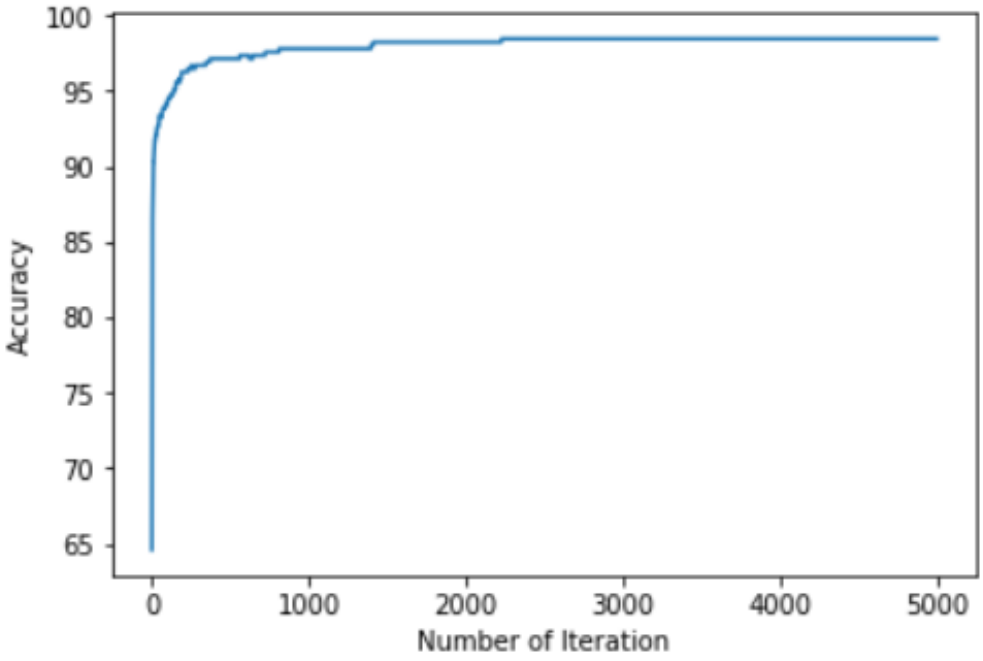

Figure 8

The validation dataset samples have been tested with 3 different learning rates for 5000 epochs. This was performed in order to tune out the best fitting hyperparameters in order to obtain best performance. Following are the observations made:

1. **For learning rate = 0.1**

   Cost = 0.127455
   Accuracy = 98.2561403508773 %
   True Positive: 16
   True Negative: 40
   False Negative: 0
   False Positive: 1
   True Positive Rate / Recall: 100.00%
   Precision: 94.12%

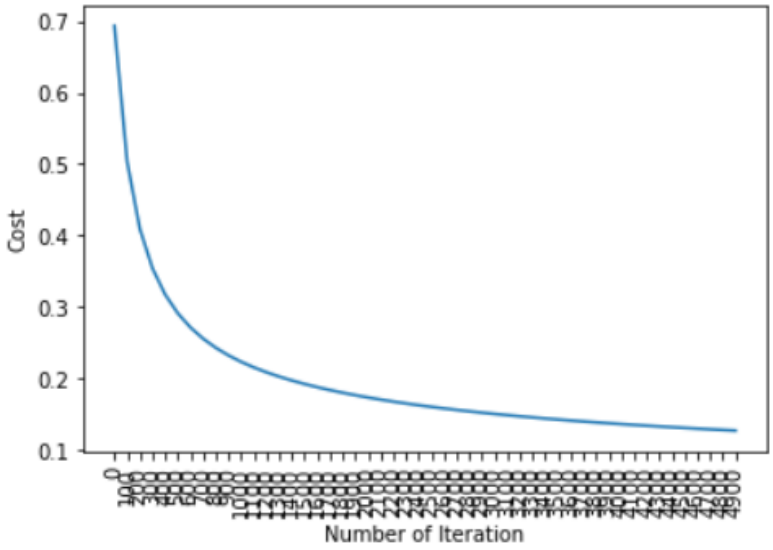

Figure 9 : Cost vs epochs plot for learning rate 0.1

2. For learning rate = 0.5

Cost = 0.079245
Accuracy: 98.24561403508773 %
True Positive: 16
True Negative: 40
False Negative: 0
False Positive: 1
True Positive Rate / Recall: 100.00%
Precision: 94.12%
False Positive Rate / Fallout: 2.44%

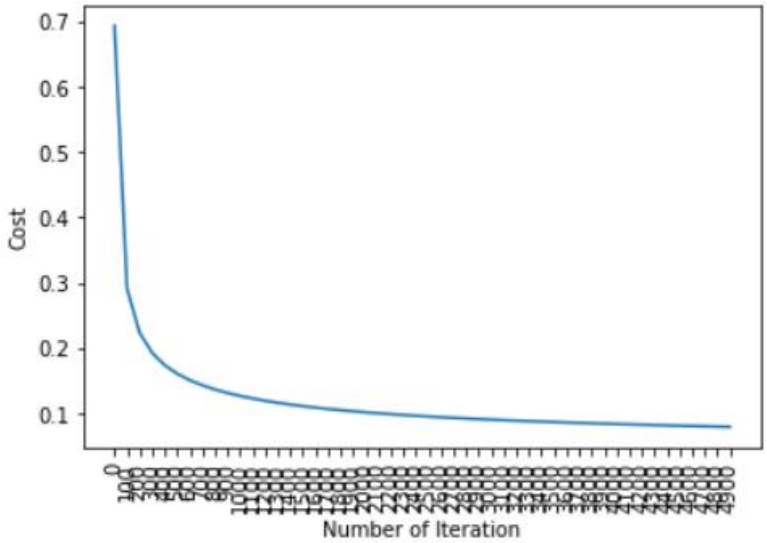

Figure 10 : Cost vs epochs plot for learning rate 0.5

3. **For learning rate = 1**

Cost = 0.067000
Accuracy = 98.24561403508773 %
True Positive: 16
True Negative: 40
False Negative: 0
False Positive: 1
True Positive Rate / Recall: 100.00%
Precision: 94.12%
False Positive Rate / Fallout: 2.44%

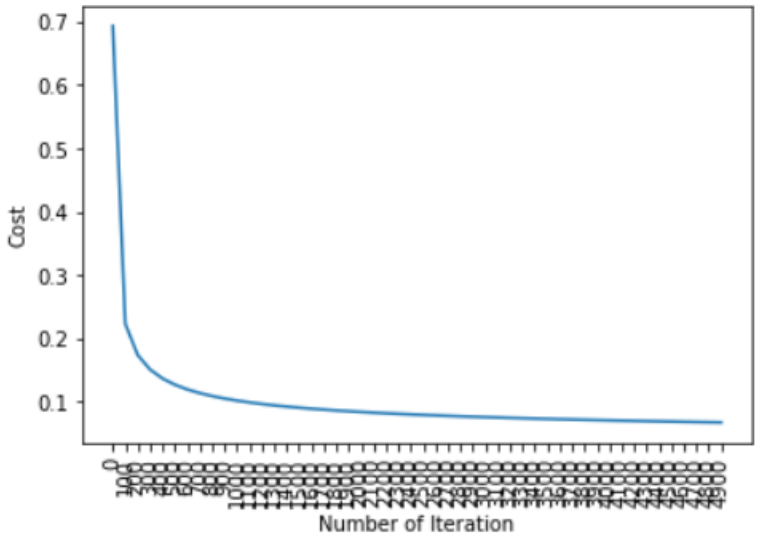

Figure 11 Cost vs Epochs plot for learning rate 1

From the above observations, it can be concluded that for learning rate = 1, the model gives its most effective performance with the least cost.

Fixing the same parameters, the testing dataset was passed into the designed LR model and the following results were collected.

Accuracy = 96.49122807017544 %
True Positive: 25
True Negative: 30
False Negative: 2
False Positive: 0
True Positive Rate / Recall: 92.59%
Precision: 100.00%
False Positive Rate / Fallout: 0.00%

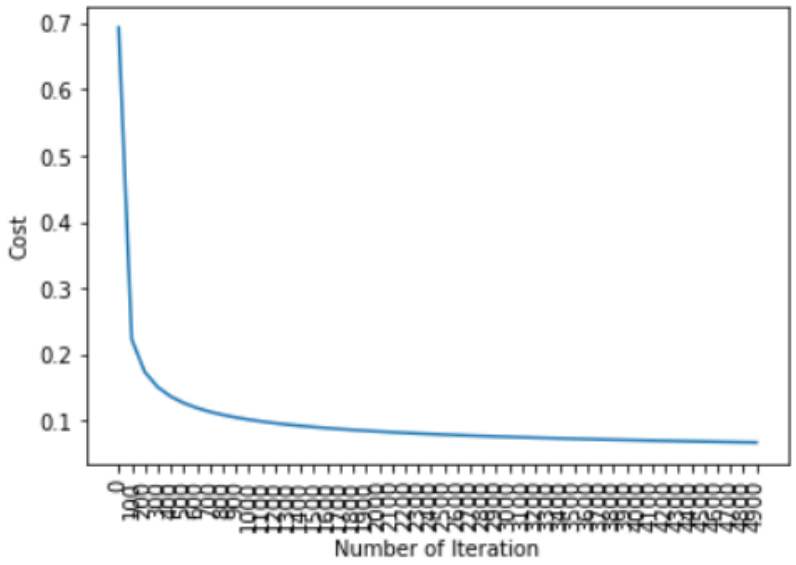

Figure 12: Cost vs epochs plot for learning rate 1 (test data)

Thus it can be concluded that the training was successful with and accuracy of approximately 98 % and, while on the testing set out it turned out to be almost 96% which is a pretty good result.

REFERENCES:

- https://medium.com/@ODSC/logistic-regression-with-python-ede39f8573c7
- https://pandas.pydata.org/pandas-docs/stable/reference/frame.html
- http://www.datasciencemadesimple.com/
- https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://stats.stackexchange.com/questions/278771/how-is-the-cost-function-from-logistic-regression-derivated