CSE574 Intro to Machine Learning Project 1

Utkarsh Behre
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
ubehre@buffalo.edu

Abstract

In this project, we were asked to perform classification using machine learning for a two-class problem on the given Wisconsin Diagnostic Breast Cancer (WDBC) dataset. We were expected to be able to classify suspected FNA cells to be benign (class 0) or malignant (class 1) using logistic regression as the classifier. After performing these tasks on the given WDBC dataset, herein lies my final report.

1 Introduction

Machine learning is an application of Artificial Intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. These systems can learn and predict an outcome with the help of several machine learning algorithms designed to manipulate the data in different ways depending on the algorithm. After all, machine learning is all about manipulating the data that we have, in order to get what we want.

1.1 Regression vs Classifier

There are many machine learning algorithms that we can use to train our system. But choosing an appropriate algorithm for our problem really depends on the problem and the data that we are dealing with. For example, are we trying to predict the price of a house? Or, perhaps we are trying to guess whether a person has breast cancer or not? Both problems clearly have a big difference to notice. In the first one we are supposed to predict how much a house may cost which could be let's say $60,000, but in the latter example we are supposed to give a yes or no answer i.e. say the person has cancer or he/she doesn't have it.

The first example discussed falls under the problem type regression, where we have to predict a value which could be anything from 0 to millions or perhaps negative values. But the other example falls under classifier problem type, where we may have 2-class or a multiclass problem. The example in this case would just be a 2-class problem, classes being having cancer and not having cancer. In these cases, we are not required to predict some value, but we are required to predict the class under which the object falls, given a set of related features. What are these features? In order to learn about what are these features, first we need to understand what a dataset would look like.

1.2 Data

In order to predict something, we need some sort of data based on whose reading we can make out our prediction. Let's take the second example that we discussed earlier i.e. to predict whether someone has breast cancer or not. In this problem, we need to have certain features of the cell nuclei present in the digitized image of a find needle aspirate (FNA) of a breast mass. These features may be things like radius, texture, perimeter, area, and so on.

Now for a system to get trained, we need some sample data which comprises of values of these features for some people, and the result i.e. whether the person has cancer or not. We call the results as labels in machine learning, which are used to tell the system whether it has made a correct or a wrong prediction for a sample.

1.3 Problem types based on data

Based on whether we have these labels at the time of the training process, or simply speaking, based on data, a problem in machine learning can be solved in 4 ways: Supervised, Unsupervised, Semi-supervised, and Reinforcement learning. In this project, we are only dealing with supervised learning where we have labels for all our sample data. The other types deal with the data having either some labels or no labels at all.

So, our focus in this project is going to be on supervised learning. More specifically we will be making use of a classifier called logistic regression in order to perform computations on the given features and figure out which class (benign i.e. class 0 or malignant i.e. class 1) a sample could possible belong to.

2 Dataset

The problem where we have all the labels available to for all the respective sample features falls under the category of supervised learning. We are given the Wisconsin Diagnostic Breast Cancer (WDBC) dataset for the purpose of this project. This dataset comprises of total 30 features, along with labels. This is clearly a supervised learning problem as we are given a total of 569 samples of features and respective labels. And since we are asked to predict whether someone has cancer or not, it clearly is a two-class classifier problem that we are dealing with here.

2.1 WDBC Dataset

The given dataset for this project is Wisconsin Diagnostic Breast Cancer (WDBC) dataset. The first and second columns of this dataset are ID and diagnosis (B/M). The remaining 30 columns are real values input features that are computed from digitized images of a fine needle aspirate of a breast mass that represent 10 different characteristics of a cell nuclei in the image.

The important part for us is to notice that we have 30 different features for 569 different images, and we know whether it was diagnosed as Benign or Malignant. The first column ID is really of no use for us. So, we have 2^{nd} column as our label in this case either B (Benign) or M (Malignant) and all the columns after that are the features that we must consider while training our model.

For this project that given dataset has been divided int 3 parts: training, validation, and testing. Training dataset consists of the 80% of the entire dataset, while validation and testing each consists of 10% of the entire dataset.

2.2 Training, Validation, Testing

The training and validation datasets are used at the time of training however the testing dataset is used at the very end to test how our model does in an unseen testing dataset. During the training, the training dataset is used to correct the system if it made a wrong prediction by penalizing it using the cost function, and validation set was used just to check how it is improved on an unused dataset. In order to observe how the model is performing on the validation set, we do find the cost for the predicted validation label as well, but we do not use this data to modify the weights.

3 Logistic Regression

For this project we will be using the classifier known as logistic regression to solve our problem. Let's now try to understand how logistic regression works.

The very first thing that we can say about these features is that all features are not that important. In other words, some features may be more important than other features in determining our output (class B/0 or M/1). This is exactly what we are supposed to find out. We call the importance of a feature as weights.

Let's say *X* represents a matrix that has all the features for all the samples available to us making it a *569x30* dimensions matrix. We need to find weights for each of those features, so we can take *w* as a 1x30 vector. There may also be some bias weight that doesn't rely on any of the given features. Let's call it w0. So, our hypothesis function z will be the dot product of X and w adding scalar bias value w0 to it. Initially the w and w0 can be initialized with anything, as they will get changed later according to the calculated error. In the project we have initiated them with zeros.

$$z = X.(w.T) + w$$

As we have dimension of X as 569 x30 and dimensions of *w transpose* vector as 30x1 we have a resulting dimension of z as 569 x 1 which will have all real valued numbers.

Now the trick is to get these numbers lie between 0 and 1 so that we can then classify them to be of class 0 or 1 based on a deciding falling value in between 0 and 1. (0.5 is considered as the deciding value for the project) In order to transform these values, we will be using a sigmoid function, whose function is to just do what we want. This function looks like below

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

Once we have this. We than need to find the cost of the error, which is basically how wrong the system was in prediction whether the class was B or M. The function for this is below

$$Loss = -\frac{1}{m}[\sum_{i=1}^{m} y^i \log(Y) + (1-y^i)\log(1-Y)]$$

Once we find the cost, then we need to update the *w* vector and the bias w0 according to a decided learning rate. This has to be done simultaneously and is done by the following equations where η represents the learning rate and Δ represents the derivative

$$w_i = w_i - \eta \Delta w_i$$

$$w_0 = w_0 - \eta \Delta w_0 \text{ , where}$$

$$\Delta w_i = -\frac{1}{m}[y - \sigma(z)]X_i$$

$$\Delta w_0 = -\frac{1}{m}[y - \sigma(z)]$$

It is not possible to get the weights right the very first time. And so we keep doing this many times and the process where we keep modifying the weights according to the cost function is known as the gradient descent. We do this till the point where values of the weights, sort of stop changing. This point is also known as the point of convergence.

Ideally, at this point one may take a set of new features, find the z, and Y using the hypothesis and the sigmoid functions mentioned above, and based on the value of Y predict the class being 0 or 1 if the value was less than 0.5 or it was greater than or equal to 0.5.

4 Experimentation

First, we defined all the basic functions that we are going to need, such as data partitioning, sigmoid, logistic regression, prediction. Then, based on the requirements of the project we also defined some metrics evaluation related functions such as functions to find out TP, TN, FP, FN, accuracy, precision, and recall. And lastly, as needed to see the graphs for how our

model is performing the functions to plot the desired graphs were defined.

While writing above functions I ended up performing lot of unit testing just to make sure It is working in the expected way. But after we defined all these functions in the program, we used them to train and evaluate our model's performance.

4.1 Normalization

Upon observing the dataset given to us, we found out that the values of some of the features were in high range while others were on low range. For example, the feature represented in column 5 are in range 143.5 to 2501. While the feature represented in column 6 are in range 0.05262 to 0.1634. This was easy to find using numpy amin and amax functions.

This high variance in range between different features would make learning the weights harder for the system. So, to make the learning smooth and easier on our system we normalized our data.

For normalization we simply converted a feature value to its relative position between 0 and 1 when compared to the other values of the same feature. For example, in column 5 143.5 would get converted to 0 and 2501 would get converted to 1 and the rest of the values would lie somewhere in between 0 and 1. This was done using a simple formula written below.

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

4.2 Initialization

At the time of defining and writing code there were certain parameters that we had to initialize first. We initialized the weight vector using the numpy np.ones() to fill all weights as 1s and we took 1 as the initial bias as well. Then for starting point we took 1000 epochs and 0.0001 as our learning rate.

4.3 Tuning the hyperparameters

First, I started with the initialized values as mentioned in 3.2 section. The First attempt gave a very bad training loss graph with a tilted straight line going down when epochs increased. The graph is shown in the Figure 1(i). Then I tried increasing the learning rate to 0.001, but the graph looked the same. After increasing it to 0.01, I got the graph as shown in the Figure 1(ii).

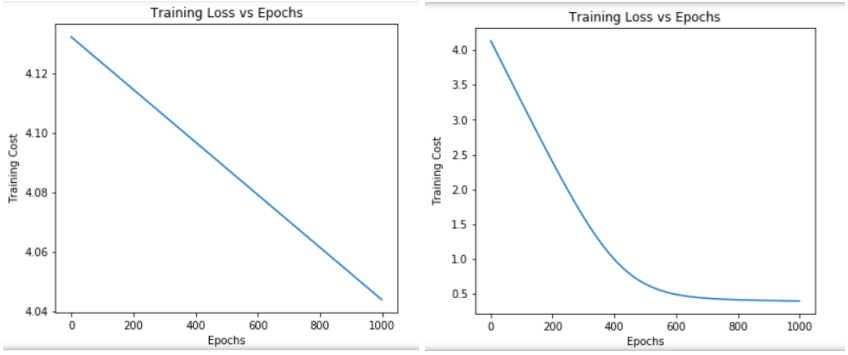

Figure 1(i) and (ii): Training loss with start w =1s, and learning rate (i) 0.0001 (ii) 0.01

Even though it looks like the error is minimized somewhat minimized, but the graph did not look like it is converging to a low point. I tried changing epochs to 500 and learning rates to 0.1 and tried few more combinations, but the loss graph didn't make for a perfect even converging plot.

So, instead of further modifying epochs and learning rate, I decided to change the initial weight values and w0 to zeros using np.zeros instead of 1s. I set the epochs to 1000 and

learning rate to 0.01 to see how it performed. This didn't look that great, but at least, I could see the graph converging just slightly but evenly at this point instead of a straight line. This is shown in the Figure 2(i). Then, I just increased the learning rate to 0.1 which gave a fast converging graph with high accuracy of about 96.49%. Training loss for this graph is shown in the Figure 2(ii).

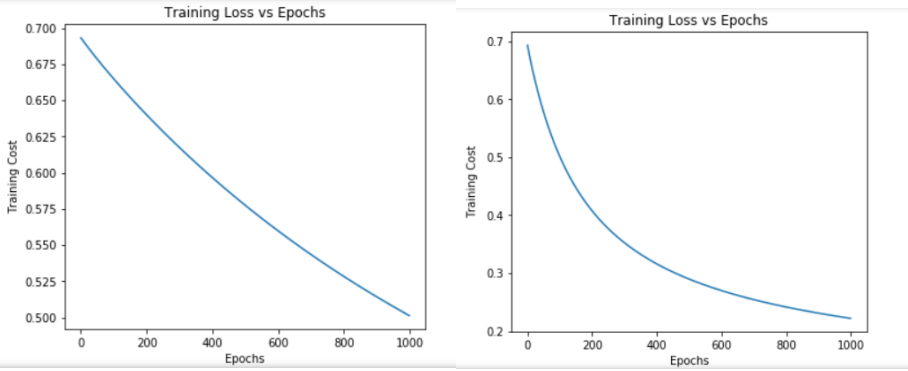

Figure 2 (i) and (ii): Training Loss with start w = 0s and learning rate (i) 0.01 (ii) 0.1

Since this graph was very close to what was expected, I made some final adjustments to epochs and learning rates by and started getting a good converging graph around 3000 epochs and .07 learning rate. The final graph I got is shown in Figure 3.

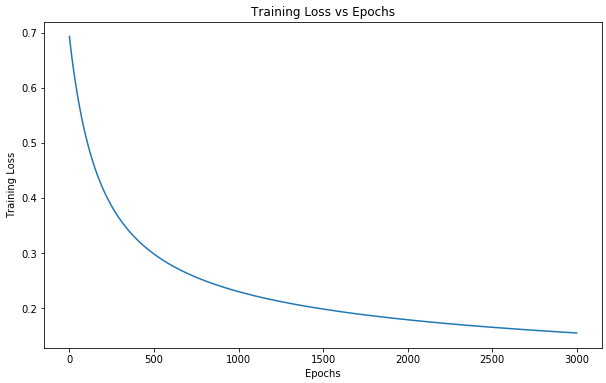

Figure 3: Training Loss with start w = 0s, epochs = 3000 and learning rate = 0.9

A comparison graph between training loss and validation loss is given below for the final parameters.

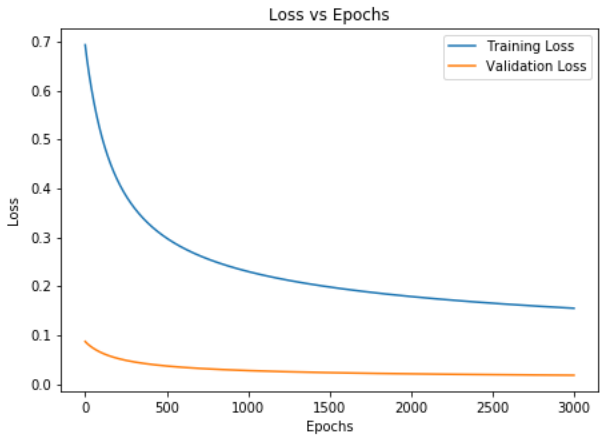

Figure 4: Training and Validation Loss vs Epochs (learning rate = 0.9)

Apart from the graphs, below is a table 1 with the recorded values of accuracy, precision, and recall that I got for different tuning parameters.

Table 1: Hyper parameters and respective accuracy, precision, and recall values

Initial w, w0 values	Epoch	Learning Rate	Accuracy (in %age)	Precision (in %age)	Recall (in %age)
1s, 1	1000	.0001	33.33	33.33	100
1s, 1	1000	.001	28.07	28.07	100
1s, 1	2000	.001	28.07	28.07	100
1s, 1	1000	.01	96.49	93.75	93.75
1s, 1	500	.01	57.89	40	100
0s, 0	1000	.01	91.23	100	78.26
0s, 0	1000	.1	96.49	100	91.3
0s, 0	800	.01	91.23	100	75
0s, 0	800	.07	92.98	94.44	85.00
0s, 0	10000	.09	94.74	100	88.46
0s, 0	3000	.09	92.98	100	84.62

5 Results

All the graphs for this final setting of initial weights 0, epochs 800 and learning rate .07 are given in the Figures 3, 4, 5, 6, and 7. These show training loss only (Figure 3), training and validation loss (Figure 4), training and validation accuracy (Figure 5), training and validation precision (Figure 6), and lastly training and validation recall (Figure 7). All these values are plotted against epochs.

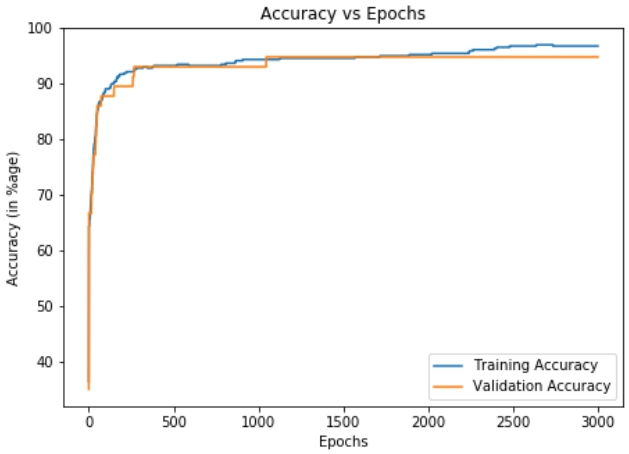

Figure 5: Training Accuracy vs Epochs and Validation Accuracy vs Epochs

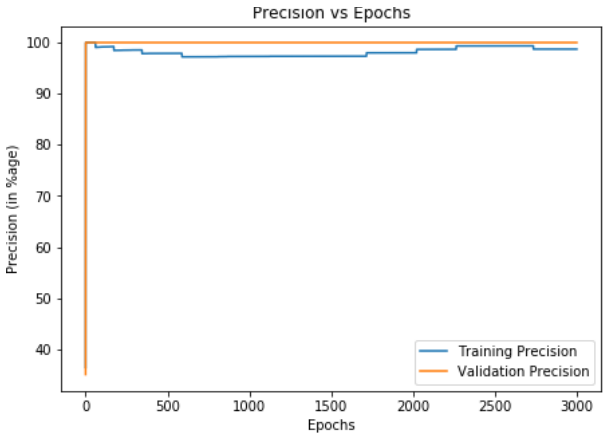

Figure 6: Training Precision vs Epochs and Validation Precision vs Epochs

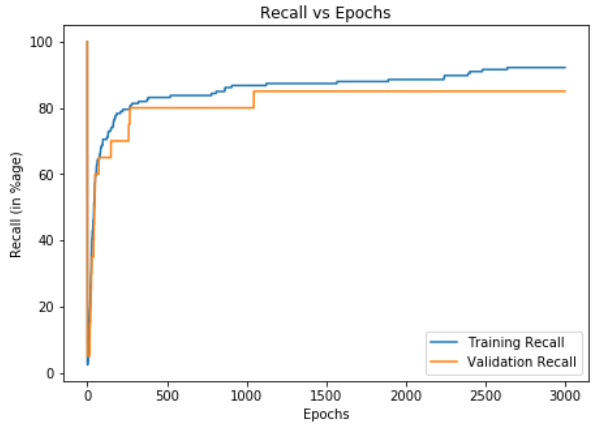

Figure 7: Training Recall Vs Epochs and Validation Recall vs Epochs

After this I ran the prediction for the remaining test dataset and got accuracy as 92.98, precision as 94.44, and recall at around 85%, which felt like a good stopping point considering accuracy wasn't way too high which would be overfitting case and it wasn't too low which would be underfitting and the precision and recall values seemed to be acceptable.

Table 2: Final Hyper parameters and the results

Initial w, w0 values	**Epoch**	**Learning Rate**	**Accuracy (in %age)**	**Precision (in %age)**	**Recall (in %age)**
0s, 0	3000	.09	92.98	100	84.62

Lastly the final weights in the w vector for all 30 features is plotted as a bar graph and is shown in the Figure 8. This makes a clear display of the importance of all the features in predicting cancer.

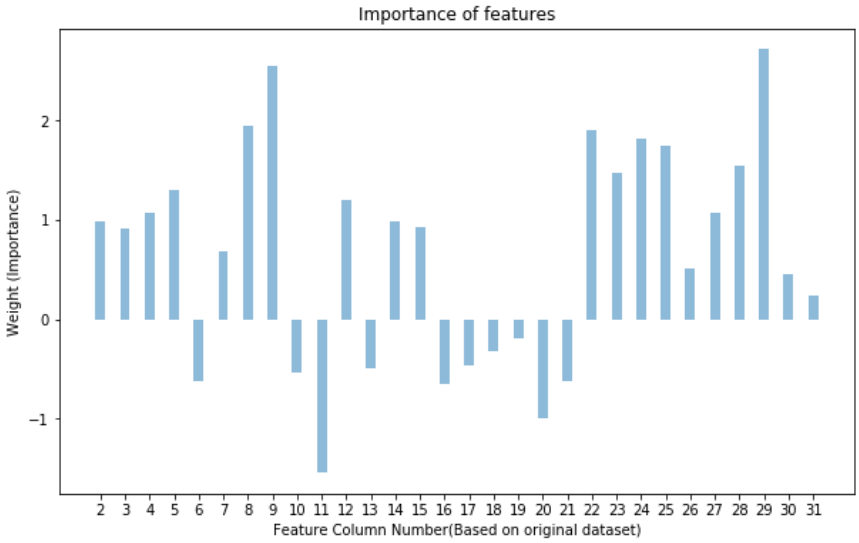

Figure 8: Final values of weights against the respective features

6 Conclusion

We were able to design the model from scratch for predicting presence of cancer. And after fine-tuning the different parameters such as learning rate, epochs, initial weight and bias values, we came to a final set of values that made our linear regression classifier good enough for predicting presence of cancer for an unseen set of features.

Acknowledgments

I am grateful to Professor Sargur Srihari for helping me understand the concepts of machine learning and its application. I also thank Mihir Chauhan and Tiehang Duan for taking efforts in providing clarifications and better understanding of implementation of logistic regression in python.

References

[1] https://stackoverflow.com/questions/29576430/shuffle-dataframe-rows

[2]https://ublearns.buffalo.edu/bbcswebdav/pid-5107775-dt-content-rid-25436584_1/courses/2199_23170_COMB/1.1.2%20Python%2BMLFrameworks.pdf

[3] https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplots.html

[4] https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b

[5] https://pythonspot.com/matplotlib-bar-chart/

[6] https://codeyarns.com/2014/10/27/how-to-change-size-of-matplotlib-plot/

[7] https://www.researchgate.net/publication/303326261_Machine_Learning_Project