Predict Diabetes using Perceptron

Rishabh Kumar Khanagwal
University of Adelaide
University of Adelaide,SA,North Terrace,5000
rishabhkumar.khanagwal@student.adelaide.edu.au

Abstract

A perceptron is a densely connected neural network. Perceptron is a type of implementation for Linear Classification, here we will be using perceptron algorithm for binary linear classification. We will be implementing a single layer perceptron. It is always a good practice to visualize the data so we visualized the raw data. We visualized the data using many functions like plotting histograms, boxplots, correlation matrix and after getting the feel of the data we started playing with the perceptron algorithm. We will be following the common workflow of Machine Learning. We divided the data into three parts that is Training, Validation and Testing. As this is a linear model, so we will be using the equation of straight line, which is y=mx+c. We will be using some formulas with which we will be updating weights, calculating loss values,getting prediction ,accuracy. We will also store the best weights after our 'n'epochs, in this case of study we used 20 epochs, and we will use these best weights on the testing dataset and after doing that we get an accuracy of 74.29%.After that we consider perceptron from sklearn library and see that our accuracy was 69.26%.

1. Introduction

Perceptron is a mathematical model for biological neuron. In biological neuron it receives electric signals as inputs and produces some outcome either in a form of electrical signals or physical movement [3]. Perceptrons are also referred as dense layer neural network. Here in this paper we will be using single layer perceptron. Perceptron provides an easy way for binary classification [1]. We usually use perceptron for supervised learning of binary classifiers. We are performing classification on Indian diabetes dataset. We will try to predict about a person who either has diabetes or not, based on the architecture of perceptron. It is simple we have inputs which are all the features of the dataset ,we perform dot product of those features with a specific weights. After that we perform some mathematical operations to get the binary results.

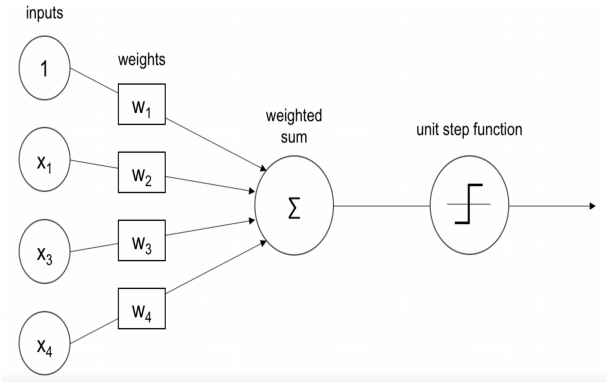

Figure 1. Architecture of Perceptron. [5]

2. Methodology

Here we are implementing a single layer perceptron. As per our dataset we have eight columns and one label column, we know that for binary classification the we use the same simple equation of straight line which is y=mx+c, where m is the slope of the line and c is the constant term. The interpretation of 'c' is that if its value is 0 then the line will always pass through origin. The same equation can be interpreted in terms of computer science as f(x)=b0+b1*x1 + b2*x2... Here 'bn' is the weights of the line or in lay men terms it is the slope of the line and 'b0' is the constant term or the bias.

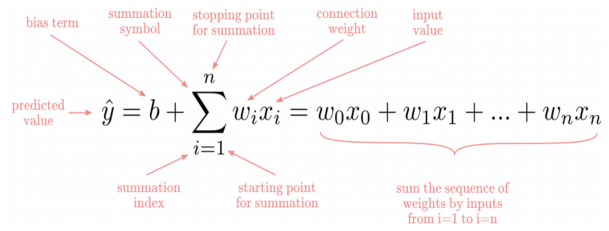

Figure 2. Simple Equation of a line. [6]

Initially we will take a bunch of random weights and fit our model with these parameter, since the weights are ran-

dom so there will be many misclassifications so we will use some formula on how to update weights and run our model again with the updated weights. We keep on running the model again and again till we get a reasonable accuracy. This is how we will be getting the best value of slope. But here the important thing to note is that how will we able to manage bias. A bias is another fancy name for the the the constant 'c' which we talked about. If we do not consider the case of bias, then it means that bias has a value equal to zero , which means our hyperplane will never deviate form the origin. [2] According to me this might not be the best case because there may be many situation in which after running the model the best acquired bias will not be zero but will be something else. Therefore I think it would not be a good case not to consider bias which is keeping the bias Zero all the time, an alternative approach can be made by initialising the bias with zero and updating it with 'n'-epochs. There may be many ways on how to get the best value of the bias. Here my approach towards finding the best weights and bias is that I first added a column containing all ones with the other features. The purpose of doing that we will be using the equation which we talked about earlier. As the bias is a single value in the classification line. Now we have one more columns so we need to one more random weight that is we now have nine columns so we will initially take nine random weights and according to the equation we have to multiply the weights with features so the value of our bias column is 1 so the first value in the array of random weights will be multiplied by 1 and it will be our bias for that particular row and the rest of the values of that row that is the column values will be multiplied by rest of the weights and this is how we get the values for the equation of a line for that row. Similarly we do this for all the rows and create a hyperplane. As we are using random weights, it is obvious to get many misclassification. So we use the weight updation formula to get better weights and bias and we can run it an appropriate number of times to get better results. It is import to store the best weights so that we can use our best acquired weights on the testing data. An 'epoch' means the number of times we run our model to get better result. An important thing to note is that we our number of epochs can be very very large because it will always update and give us the better result, but if the epochs are very large then we will be over-fitting our model. Selecting number of epochs will be different for all cases a good way can be to plot a graph of training error and validation error and deciding at which point they have the minimum values. Or the number of epochs can be 3 times the number of columns [4]. After all this it is import to consider the loss function , that is how much error you get in training and validation data, this will help you to see if your model is over-fitting or under-fitting. There can be different ways on how to calculate loss function, here in this paper we will be use the Zero-one loss to compute loss. After predicting the prediction values we can check the accuracy or any other matrix to see how well our model is.

2.1. Weights Updation

for t = 1 to T

$$w_{t+1} = w_t + \eta \sum y_i x_i 1_{\{y_i < x_i w_t > < 0\}} \tag{1}$$

t = Current number in sequence of epoch
T = total number of epochs
w_t = current weights
w_{t+1} = new updated weight
η = learning rate
y_i = label of i^{th} sample
x_i = feature value for the i^{th} sample [8]

There may be various formulas on how to update the weights. Here in this paper I will be using the above mentioned formula. This formula is very easy to interpret. First calculate the dot product of features and weights and multiply it with the label value of that row and if the result is less than zero then the result becomes one otherwise it is zero. Now multiply this result (either Zero or One) with the multiplicative result of labels and features for all the rows and columns then multiply the outcome with the learning rate and then add the previous used weights. And the output will be the new list be better weights.

2.2. Loss Function

There may be may ways to calculate loss function, here in this paper I am using Zero-one loss function because the formula used in case of updating the weights uses Zero-one method, where we either use 0 or 1 value which is multiplied with yi*xi. The 1 in the formula is an indicator function which = 1 when true otherwise 0 [8].

2.3. Prediction

We will calculate the dot product of features and weights and if the resultant is less than 0 then the prediction for that row will be -1 else 1

3. Experimental Analysis

As we discussed earlier how we are updating the weights, we also created a function that will store the best weights obtained for 'n' epochs and we will be using those acquired best weights on the testing data. After 20 epoch we get a training accuracy of 76.16% and validation accuracy of 72.04%. But it is important to note that these are the are values obtained on the last epoch. We have a function created which stores the best weights among all the epochs and we use that to get test accuracy, the validation accuracy with best weights is 73.29% and the test accuracy with the

same weights is 74.29%. As we can see that we get similar training, validation and testing accuracy so we say that our model works fine. The results demonstrate that our model is neither overfitting not underfitting. We plotted loss and accuracy curves for training and validation dataset. With Accuracy curve plot we can only conclude that overall the accuracy increases but the graph may be different if we use another random state. We may see fluctuations in accuracies with the change in random state.

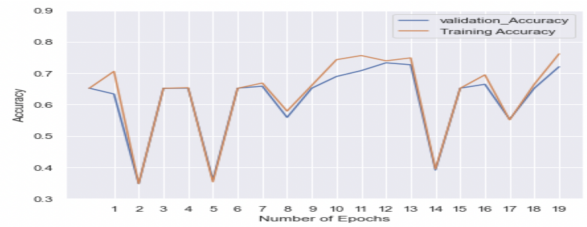

Figure 3. Accuracy Plot

Similarily we see that loss curve plot overall the loss decreases but the nature of the plot again depends on random state.

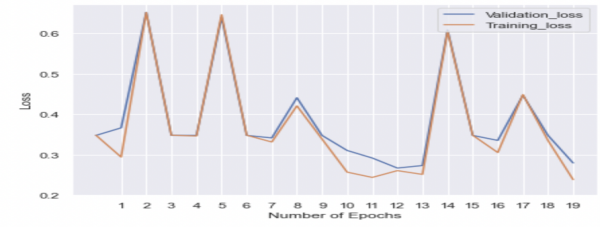

Figure 4. Loss Plot

The reason for this is that we are only implementing a single layer perceptron. Our model is too simple to give accurate result also the dataset available is not that big to get a solid outcome. To make sure that our model is working fine we imported perceptron from sklearn here we split the data into two parts that is the training and the testing part and we see that we get the best accuracy on testing data as 69.26% which is similar to our model, obviously we can increase the accuracy by optimizing hyper-parameter by using grid search or stuff like that but we are not concerned with that as we just wanted a rough idea about how well our model is in comparison to built-in model. And we are happy to see that our model works well.

4. Code

The link to the code section can be found here:
https://github.com/rishabhk2/IndianDiabetes.git

	precision	recall	f1-score	support
-1.0	0.61	0.80	0.69	25
1.0	0.86	0.71	0.78	45
accuracy			0.74	70
macro avg	0.74	0.76	0.74	70
weighted avg	0.77	0.74	0.75	70

Figure 5. Classification report for the Perceptron Model that we generated.

	precision	recall	f1-score	support
-1.0	0.60	0.38	0.47	81
1.0	0.72	0.86	0.78	150
accuracy			0.69	231
macro avg	0.66	0.62	0.63	231
weighted avg	0.68	0.69	0.67	231

Figure 6. Classification report for the Perceptron Model from sklearn

5. Conclusion

We successfully implemented perceptron. Here we learned how to plot a classification line between the two classes in a nine-dimensional plane. We understood the mechanism behind how a dense layer neural network works. We came to a conclusion why a model should always have a bias term. We applied the complex mathematical formulas and came to a conclusion where our training accuracy is 76.16% , validation accuracy is 73.29% and testing accuracy is 74.29%. We also used sklearn to compare with our model. With the default values in perceptron model of sklearn we get an accuracy of 69.26% on the test data. To improve the method we could use the improved version of perceptron called Adaline. In Perceptron we use the original class labels to update weights of the model. Adaline model uses continuous predicted values from the net input to update the weights which is a much better choice as it tells us how much were we right/wrong [7]. Also the model can be improved by implementing multi layer perceptron. As this is a single layer perceptron we get an accuracy of 74.29%. We can create a more complex model and perform grid search to find the best hyperparameter and use those values to run our model. We can also take an expert opinion on which approach to follow as we have multiple ways of implementing the perceptron also what should be a good train test split as this comes from experience. The important thing to note here is the confusion matrix that we got from out developed model and the model from perceptron.

As this is a medical dataset so I would say that 19% is a big value to get into the category of False Negative. So I

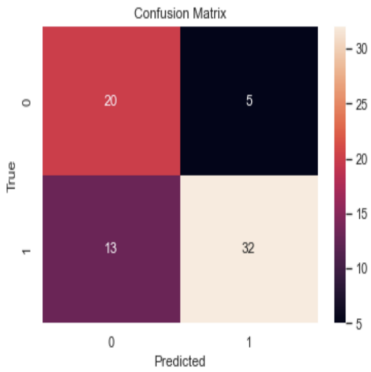

Figure 7. Confusion Matrix for the model we created

will not be too confident in using perceptron in this case.

References

[1] https://deepai.org/machine-learning-glossary-and-terms/perceptron 1

[2] https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks 2

[3] https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html 1

[4] https://gretel.ai/gretel-synthetics-faqs/how-many-epochs-should-i-train-my-model-with 2

[5] https://www.oreilly.com/library/view/deep-learning-for/9781788295628/33d24719-7e83-41c1-983e-59310eacdd50.xhtml 1

[6] https://pabloinsente.github.io/the-adaline 1

[7] https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html 3

[8] https://myuni.adelaide.edu.au/courses/79140/pages/module-2-online-learning?module$_item_id = 2838423$2