Available online at: http://www.ijmtst.com/vol6issue02.html

International Journal for Modern Trends in Science and Technology

ISSN: 2455-3778 :: Volume: 06, Issue No: 02, February 2020

Steps to Classify the Breast Cancer type accuracy using Neural Network – AI Machine Learning

Subash Kumar

Bachelor of Engineering Computer Science, Anna University,

To Cite this Article

Subash Kumar, "Steps to Classify the Breast Cancer type accuracy using Neural Network – AI Machine Learning", *International Journal for Modern Trends in Science and Technology*, Vol. 06, Issue 02, February 2020, pp.-62-65.

Article Info

Received on 21-January-2020, Revised on 05-February-2020, Accepted on 11-February-2020, Published on 19-February-2020.

ABSTRACT

A timely diagnosis of any disease is critical in the medical field, with an increasing population of breast cancer patients, this paper is dedicated to all medical professionals who are trying to save many lives. Machine learning algorithms such as support vector machines will help physicians to diagnose accurately, a family of algorithms known as neural networks has recently seen a revival under the name "deep learning." While deep learning shows great promise in many machine learning applications, deep learning algorithms are often tailored very carefully to a specific use case. Here, we will only discuss some relatively simple methods, namely multilayer perceptrons for classification and regression, which can serve as a starting point for more involved deep learning methods. Multilayer perceptrons (MLPs) are also known as (vanilla) feed-forward neural networks, or sometimes just neural networks.

KEYWORDS: *Machine Learning, Deep Learning, Artificial Intelligence, Medicine, Breast Cancer, Cancer prognosis and prediction, Data Science, Neural Network, Breast Cancer Classifier, Regression, Supervised Machine Learning*

I. INTRODUCTION

In this paper, we will use 12 attributes, out of which 10 real-valued features from each cell nucleus obtained from the Wisconsin diagnostic breast cancer dataset that explains about the stage of breast cancer M (Malignant) and B (Benign).

Features explanations:

1. ID: Patient id
2. Diagnosis (M = Malignant, B = Benign)
3. Radius(mean of distances from the center to points on the perimeter) (worst). Worst texture. Texture (standard deviation of gray-scale values) (worst). Worst perimeter. perimeter (worst)
4. Texture (Breast cancer can cause changes and inflammation in skin cells that can lead to texture changes), here we can take the standard deviation of grey scaled values
5. Perimeter: Size of the core tumor
6. Area: Area of the core tumor
7. Smoothness: Local variation in radius length
8. Compactness: (perimeter^2 / area - 1.0)
9. Concavity (severity of concave portions of the contour)
10. Concave points (Number of concave portions of the contour)
11. Symmetry
12. Fractal Dimension ("coastline approximation" -1)

Except for the ID and Diagnosis, all other features are divided into three parts, the first part is Mean that tells about the mean of all cells, the

second part is Standard Error that tells about the standard the error of the cells and the third part is the worst mean of the worst cells. Now we have a total of 30 features that will be the data for the Support Vector Machine Learning - Artificial Intelligence (Machine Learning & Deep Learning)

S.No	Features
1	radius_mean
2	texture_mean
3	perimeter_mean
4	area_mean
5	smoothness_mean
6	compactness_mean
7	concavity_mean
8	concave points_mean
9	symmetry_mean
10	fractal_dimension_mean
11	radius_se
12	texture_se
13	perimeter_se
14	area_se
15	smoothness_se
16	compactness_se
17	concavity_se
18	concave points_se
19	symmetry_se
20	fractal_dimension_se
21	radius_worst
22	texture_worst
23	perimeter_worst
24	area_worst
25	smoothness_worst
26	compactness_worst
27	concavity_worst
28	concave points_worst
29	symmetry_worst
30	fractal_dimension_worst

For example, field 1 is the Mean Radius, field 11 Standard Error Radius, 21 Worst mean Radius of the Tumor.

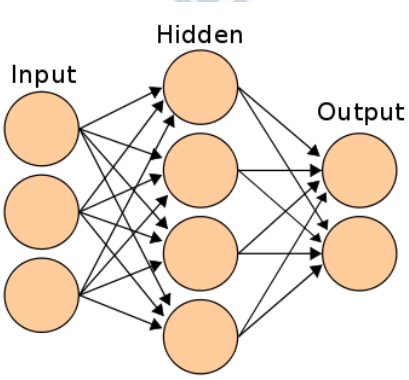
All feature values are recorded with four significant digits. Missing values are none (no missing values), class distribution 357 Benign, 212 Malignant.

II. METHODOLOGY

The Neural Network Model: Multi-Layer Perceptron can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision. The prediction by a linear regressor

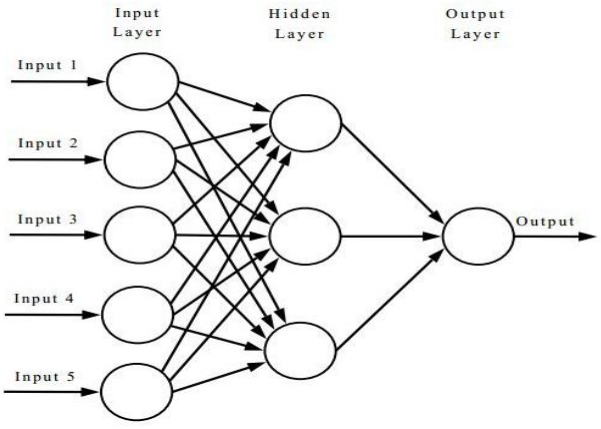
$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b$, in this equation, $\hat{y}$ is a weighted sum of the input features, x[0] to x[p], weighted by the learned coefficients w[0] to w[p]. We could visualize the equation

Formulation of Neural Network



Here, each node on the left represents an input feature, the connecting lines represent the learned coefficients, and the node on the right represents the output, which is a weighted sum of the inputs. In an MLP this process of computing weighted sums are repeated multiple times, first computing hidden units that represent an intermediate processing step, which is again combined using weighted sums to yield the final result

Simple Artificial Neural Network



By default, the MLP uses 100 hidden nodes, which is quite a lot for this small dataset. We can reduce

the number (which reduces the complexity of the model) and still get a good result

This model has a lot more coefficients (also called weights) to learn: there is one between every input and every hidden unit (which make up the hidden layer), and one between every unit in the hidden layer and the output. Computing a series of weighted sums is mathematically the same as computing just one weighted sum, so to make this model truly more powerful than a linear model, we need one extra trick. After computing a weighted sum for each hidden unit, a nonlinear function is applied to the result—usually the rectifying nonlinearity (also known as a rectified linear unit or relu) or the tangenshyperbolicus (tanh). The result of this function is then used in the weighted sum that computes the output, ŷ.

The relu cuts off values below zero, while tanh saturates to –1 for low input values and +1 for high input values. Either nonlinear function allows the neural network to learn much more complicated functions than a linear model could. For the small neural network, the full formula for computing ŷ in the case of regression would be (when using a tanh nonlinearity)

h[0] = tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3])
h[1] = tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3])
h[2] = tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3])
ŷ = v[0] * h[0] + v[1] * h[1] + v[2] * h[2]

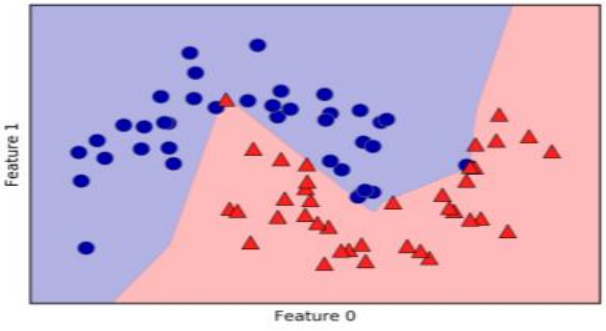
Here, w are the weights between the input x and the hidden layer h, and v is the weights between the hidden layer h and the output ŷ. The weights v and w, are learned from data, x is the input features, ŷ is the computed output, and h are intermediate computations. An important parameter that needs to be set by the user is the number of nodes in the hidden layer. This can be as small as 10 for very small or simple datasets and as big as 10,000 for very complex data. It is also possible to add additional hidden layers

Having large neural networks made up of many of these layers of computation is what inspired the term "deep learning."

Process of implementing Neural Network: Let's look into the workings of the MLP by applying the MLPClassifier to the breast cancer dataset.

Python has the inbuilt library Sklearn Neural Network with MLP Classifier which is a widely used library for implementing the machine learning algorithms. The neural network learned a very nonlinear but relatively smooth decision boundary. We used the algorithm='l-bfgs', which we will discuss later. By default, the MLP uses 100 hidden nodes, which is quite a lot for thissmall dataset. We can reduce the number (which reduces the complexity of the model) and still get a good result

Decision boundary learned by a neural network



Finally, we can also control the complexity of a neural network by using an l2 penalty to shrink the weights toward zero, as we did in ridge regression and the linear classifiers. The parameter for this in the MLPClassifier is alpha (as in the linear regression models), and it's set to a very low value (little regularization). The accuracy of the MLP is quite good, but not as good as the other models, this is likely due to the scaling of the data. Neural networks also expect all input features to vary in a similar way, and ideally to have a mean of 0 and a variance of 1. We must rescale our data so that it fulfils these requirements.

III. Related Work

One of the possible inference we can make is that features that have very small weights for all of the hidden units are "less important" to the model. We can see that "mean smoothness" and "mean compactness," in addition to the features found between "smoothness error" and "fractal dimension error," have relatively low weights compared to other features. This could mean that these are less important features or possibly that we didn't represent them in a way that the neural network could use.

IV. Methodology

Neural networks have re-emerged as state-of-the-art models in many applications of machine learning. One of their main advantages is thatthey are able to capture the information contained in large amounts of data and build incredibly complex models. Given enough computation time, data, and careful tuning of the parameters, neural networks often beat other machine learning algorithms (for classification and regression tasks). This brings us to the downsides. Neural networks—particularly the large and powerful ones—often take a long time to train. They also require careful pre-processing of the data, as we saw here. Similarly to SVMs, they work best with "homogeneous" data, where all the features have similar meanings. For data that has very different kinds of features, tree-based models might work better. Tuning neural network parameters is also an art unto itself. In our experiments, we barely scratched the surface of possible ways to adjust neural network models and how to train them. A helpful measure when thinking about the model complexity of a neural network is the number of weights or coefficients that are learned. If we have a binary classification dataset with 100 features, and you have 100 hidden units, then there are 100 * 100 = 10,000 weights between the input and the first hidden layer. There are also 100 * 1 = 100 weights between the hidden layer and the output layer, for a total of around 10,100 weights. If we add a second hidden layer with 100 hidden units, there will be another 100 * 100 = 10,000 weights from the first hidden layer to the second hidden layer, resulting in a total of 20,100 weights. If instead, we use one layer with 1,000 hidden units, you are learning 100 * 1,000 = 100,000 weights from the input to the hidden layer and 1,000 x 1 weights from the hidden layer to the output layer, for a total of 101,000. If we add a second hidden layer you add 1,000 * 1,000 = 1,000,000 weights, for a whopping total of 1,101,000—50 times larger than the model with two hidden layers of size 100. A common way to adjust parameters in a neural network is to first create a network that is large enough to over fit, making sure that the task can actually be learned by the network. Then, once you know the training data can be learned, either shrink the network or increase alpha to add regularization, which will improve generalization performance.

V. Results and Discussion

The results are much better after scaling, and already quite competitive. We got a warning from the model, though, that tells us that the maximum number of iterations has been reached. This is part of the adam algorithm for learning the model and tells us that we should increase the number of iterations. Increasing the number of iterations only increased the training set performance, not the generalization performance. Still, the model is performing quite well. As there is some gap between the training and the test performance, we might try to decrease the model's complexity to get better generalization performance. Here, we choose to increase the alpha parameter (quite aggressively, from 0.0001 to 1) to add stronger regularization of the weights.

Accuracy on training set: 0.988 Accuracy on test set: 0.972

References

[1] https://www.analyticsvidhya.com/blog/2017/09understaing-support-vector-machine-example-code/

[2] https://www.kaggle.com/gargmanish/basic-machine-learning-with-cancer

[3] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[4] https://www.kaggle.com/uciml/breast-cancer-wisconsin-data#data.csv

[5] https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/version/1#README_data_info.rtf

[6] https://ieeexplore.ieee.org/document/6044334/references#references

[7] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html