Usage of Perceptron Model to Predict Diabetes Among Pima Indians

October 20, 2024

Abstract

This paper explores the use of the perceptron model to predict diabetes in the Pima Indian population in North America. By using various features such as age, pregnancies, glucose level, and more, the perceptron model is trained, and performance metrics are evaluated across different variations of regularization and activation functions.

1 Introduction

With the help of algorithms, we can allow computers to do tasks more efficiently compared to humans. Algorithms can range from performing tasks quicker, such as sorting data structures with efficiency and speed. The problem with static algorithms is that they often require constant human intervention. For example, we can create an algorithm to check, based on certain criteria, whether a file contains sensitive information or not. This includes checking through the algorithm if there are any fields called "name", "phone number", etc. But there could be instances where we are dealing with thousands of files where the fields may not be specified correctly, and we may encounter personal information that should not be made public. Therefore, training the algorithm based on data becomes crucial for scaling up operations. We want the computer to automatically check and file information with less human intervention (almost perfectly). This is where artificial intelligence comes into play, allowing the computer to train algorithms to adjust themselves based on the available data.

Artificial intelligence capabilities have grown exponentially in recent years; from simple computing models built in the 1950s to transformers now powering large language models.

2 Basics of Artificial Neural Networks

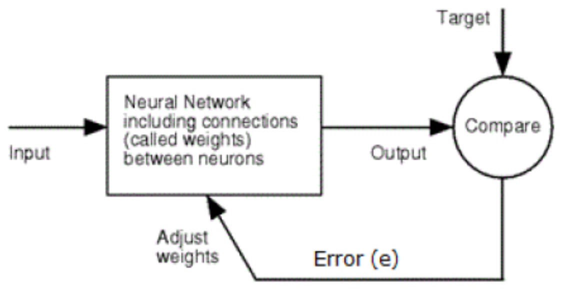
Target
Input
Neural Network including connections (called weights) between neurons
Output
Compare
Adjust weights
Error (e)

Figure 1: Simple Neural Network

A neural network is a model structure with an algorithm for fitting the model to a given dataset using generic nonlinearity, allowing the parameters to be adjusted so that it can deal with various non-linearities. Data learning through training is a methodology where the neural network is trained to work with the given problem.

As seen in the diagram below, an input enters the neuron of the neural network, and an output is generated. When we train the data, we compare the output generated by the model to the actual data. The difference, called the error, is used as a training signal for the model.

3 Perceptron Model

The perceptron is the basic processing element and has inputs that may come from the training dataset or may be the outputs of other perceptrons. Associated with each input, $x_j \in \mathbb{R}$, $j = 1, \dots, d$, is a connection weight, $w_j \in \mathbb{R}$, and the output, y, in the simplest case is a weighted sum of the inputs.

$$y = \sum_{j=1}^{d} x_j w_j + w_0 \qquad \text{......................}(1)$$

w_0 is the intercept value to make the model more general; it is generally modeled as the weight coming from an extra bias unit, x_0, which is always +1. We can write the output of the perceptron as a dot product:

$$y = w^T x \qquad \text{...}(2)$$

Where $w = [w_0, w_1, \dots, w_d]^T$ and $x = [1, x_1, \dots, x_d]^T$ are augmented vectors that also include the bias weight and input.

The perceptron as defined in equation (1) defines a hyperplane and as such can be used to divide the input space into two: the half-space where it is positive and the half-space where it is negative.

4 Pima Indians Diabetes Dataset

For our study, we will be trying to create a model that will allow us to predict diabetes among the Pima Indian population in North America based on relevant features such as age, pregnancies, glucose level, blood pressure, skin thickness, BMI, and insulin. The dataset has been scaled to aid in the model building for our report and is collected by UCI. During the data sanity check, it was identified that there were nine null values in the rows of the study population. I have decided to eliminate these rows from the dataset to assist in model building.

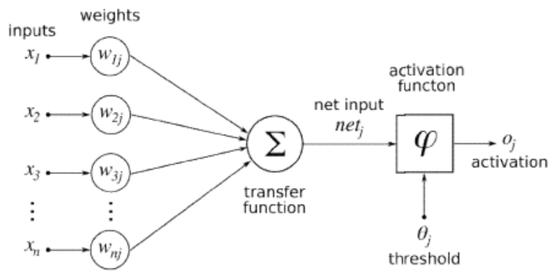

Figure 2: Perceptron Model

5 Building the Perceptron Model in Python

The roadmap for building the perceptron model was as follows:

- **Prepare Data:** Load and split the dataset into features and outcomes.
- **Initialize Model:** Define parameters like learning rate and number of iterations.
- **Define Activation Function:** Use a step function to classify the predictions.
- **Train the Model:** Update weights and bias based on prediction errors by training on the train data.
- **Predict:** Use the learned weights and bias to make predictions on the test data.
- **Evaluate:** Measure model accuracy by comparing predictions across various models and finalize the best available.

6 Base Perceptron Model

The baseline model was built to understand how the model performs in predicting diabetes outcomes without any parameter tweaks. The following steps outline the construction of the model in Python:

6.1 Class Initialization:

- **Learning rate:** Controls how much the weights are updated during training. A smaller learning rate means smaller weight updates.
- **Epochs:** Number of iterations the model will go through the training data.
- **Weights:** The weight vector is initialized later to match the number of features in the data.
- **Bias:** A constant added to the weighted sum of inputs, initialized to zero.

6.2 Step Function:

- This is the activation function. The perceptron uses a simple step function to classify the input.
- If the input is greater than or equal to zero, it predicts 1. If the input is less than zero, it predicts -1.

6.3 Data Dimensions:

- The input data X is an array with a set of rows (number of data points) and columns (number of input features).
- **Initialize weights and bias:** The weights are initialized to zeros, and the bias is set to zero as well.

6.4 Training the Model:

- **Forward Pass:** For each sample, calculate the linear output as the dot product of input features and weights plus the bias.
- **Prediction:** Apply the activation function to obtain the predicted class.
- **Weight Update Rule:** If the prediction is wrong, update the weights and bias using the following rule:

7 Limitations on the Perceptron Model

- Classifying non-binary and non- linear problems: The single layer perceptron model is tuned to work only on linearly separable data. This will not help in the convergence to ideal solutions as the separation with clear distinction is not possible as binary class is inefficient. Also this causes to not have a probabilistic output.
- Model sensitivity: The initial weights chosen have to be carefully selected as they have an influence on how the convergence to solution can be optimally achieved.

8 Fine Tuning the Perceptron Model

Now that we have a standard single-layer perceptron model with the step-wise activation function, we will now investigate ways to enhance the existing model to work on the dataset for accurate predictions effectively.

8.1 Activation Functions

We have used various activation functions to see how the model can be enhanced to increase the accuracy of the predictions. The three activation functions utilized are the stepwise function, ReLU, and Sigmoid function. Below, I detail the reasoning behind experimenting with these functions in the model.

8.1.1 Stepwise Activation Function

$$\text{step}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

The basic perceptron model implemented has the stepwise function as its activation function.

This produces an output of either 1 or -1, classifying inputs in a binary manner. However, since the function is non-differentiable, it limits the types of optimization techniques that can be applied.

8.1.2 ReLU (Rectified Linear Unit) Activation Function

The ReLU activation function outputs the input directly if it is positive, and zero for negative inputs, with a range from 0 to infinity. ReLU is popular in modern neural networks due to its simplicity and efficiency. In this experiment, ReLU was used with hinge loss. The accuracy was moderate, as ReLU is more suitable for deeper networks than the shallow perceptron model.

8.1.3 Sigmoid Activation Function

Equation:

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

Range: (0, 1)

The output of the sigmoid function ranges between 0 and 1, making it suitable for binary classification by converting input into a probability-like output. This is ideal for the study as we are dealing with binary classification for diabetes prediction. In our model with Sigmoid Activation, Hinge Loss, L1/L2 Regularization, the combination produced decent results, especially with L1 regularization. In another model, Sigmoid Activation, Squared Loss, the squared loss function performed reasonably well but did not outperform the hinge loss models.

8.2 Loss Functions

Two loss functions were used in the experiments—hinge loss and squared loss. The purpose of these functions is to measure the distance between predicted output and actual output, aiding in the model's learning process.

8.2.1 Hinge Loss Function

Formula:

$$L_{\text{hinge}}(y, f(x)) = \max(0, 1 - y \cdot f(x))$$

where y is the true label (1 or -1) and $f(x)$ is the predicted output.

Hinge loss is commonly used in binary classification and works to create a decision boundary with a margin separating the data points. It increases linearly when predictions are incorrect or too close to the boundary. In the model Sigmoid Activation, Hinge Loss, L1/L2 Regularization, hinge loss worked well, particularly with L1 regularization. ReLU Activation, Hinge Loss, was also tested, though performance was lower.

8.2.2 Squared Loss (MSE)

Squared loss penalizes the square of the difference between predicted and actual values. While commonly used for regression tasks, it was also applied in classification tasks for the perceptron model. In the model Sigmoid Activation, Squared Loss, the sigmoid activation with squared loss performed moderately well but did not surpass the hinge loss models.

Formula: $\mathrm{L}_{\text{squared}}(y, \hat{y}) = (y - \hat{y})^2$

8.3 Regularization Techniques

Two regularization methods were applied in the study—L1 and L2 regularization.

8.3.1 L1 Regularization (Lasso)

L1 regularization performed well with hinge loss and sigmoid activation. By forcing less important weights to zero, it created simpler models that focused on the most relevant features. The best-performing model combined sigmoid activation, hinge loss, and L1 regularization, producing a well-defined decision boundary that worked effectively for binary classification.

8.3.2 L2 Regularization (Ridge)

L2 regularization was less effective than L1 regularization. While it helped prevent extreme weight values and reduced overfitting, it did not provide the same level of sparsity or performance as L1 regularization.

8.3.3 Final Model

Based on the model metrics, such as accuracy, F1-score, precision, and recall, the choice of the perceptron model for predicting the diabetes outcome is the perceptron model with the sigmoid activation function, hinge loss, and L1 regularization. The sigmoid activation smooths the outputs, providing a probability-like score for binary classification. The hinge loss encourages the model to separate the two classes with a large margin, ensuring better generalization and robustness. The L1 regularization reduces overfitting by simplifying the model and driving irrelevant weights to zero, making the model more interpretable and robust.

- **Perfect Recall (1.00):** The model is excellent at identifying all positive cases (i.e., it doesn't miss any positive examples).
- **Moderate Precision (0.65):** The model sometimes predicts false positives (i.e., it predicts positive for some negative examples), but it's still reasonably accurate in identifying true positives.
- **Good F1-Score (0.79):** This suggests a good balance between precision and recall, making this model well-suited for cases where both false negatives (missed positives) and false positives (incorrect positives) are important.
- **Accuracy (0.65):** The overall accuracy is moderate but the best among perceptron models, indicating that while the model captures most of the positive cases, its overall performance across both positive and negative classes is not perfect.

Model	**F1-Score**
Sigmoid Activation, Hinge Loss, L1 Regularization	0.790438
Sigmoid Activation, Hinge Loss, L2 Regularization	0.426584
Sigmoid Activation, Learning Rate Decay	0.503254
Sigmoid Activation, Momentum, Hinge Loss, L1 Regul.	0.533333
Basic Perceptron with Step Activation (Original)	0.501475
Sigmoid Activation, Squared Loss	0.543353
ReLU Activation, Hinge Loss	0.469512
Sigmoid Activation, Squared Loss, L1 Regularization	0.000000
Sigmoid Activation, Squared Loss, L2 Regularization	0.000000
Sigmoid Activation with Early Stopping	0.531387

Table 1: Model Performance: F1-Score

Model	**Precision**
Sigmoid Activation, Hinge Loss, L1 Regularization	0.653491
Sigmoid Activation, Hinge Loss, L2 Regularization	0.913907
Sigmoid Activation, Learning Rate Decay	0.544601
Sigmoid Activation, Momentum, Hinge Loss, L1 Regul.	0.899522
Basic Perceptron with Step Activation (Original)	0.934066
Sigmoid Activation, Squared Loss	0.959184
ReLU Activation, Hinge Loss	0.962500
Sigmoid Activation, Squared Loss, L1 Regularization	0.000000
Sigmoid Activation, Squared Loss, L2 Regularization	0.000000
Sigmoid Activation with Early Stopping	0.962963

Table 2: Model Performance: Precision

9 Github Link

https://github.com/SidharthSerjy/Deep-Learning—Diabetes-Forecast-based-on-Perceptron

References

[1] A. Acharyya, S. Dhar, and S. Bose, "Perceptron model and its application in intelligent systems," *International Journal of Development Research*, vol. 6, no. 4, pp. 7417-7420, Apr. 2016. Available online: http://www.journalijdr.com.

[2] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, pp. 386-408, 1958.

[3] R. Hecht-Nielsen, “Counterpropagation networks,” *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 25-29, 1990.

[4] H. Schmid, “Perceptron-based models for temporal sequence learning,” *Proceedings of the IEEE International Conference on Neural Networks*, pp. 525-530, 1994.

[5] C. M. Bishop, “Neural networks for pattern recognition,” *Oxford University Press*, 1995.