Perceptron Algorithm for Diabetes Prediction using PIMA Indian Dataset

Chan Ka Hing
The Univesity of Adelaide
kahing.chan@student.adelaide.edu.au

Abstract

Machine learning is a useful tool to recognise a pattern and make predictions. Perceptron is one of the common machine learning algorithms based on supervised learning. In this paper, we implement a Perceptron algorithm to predict diabetes. The model is a single-layer Perceptron model with one neuron. We demonstrate the model with a diabetes dataset and the testing accuracy is 82.5%.

1. Introduction

Diabetes is a metabolic disease characterised by high blood sugar caused by insufficient insulin [3]. In the last few decades, diabetes cases have risen dramatically worldwide [3]. Early screening of diabetes becomes increasingly important. This paper presents a Perceptron model for diabetes prediction using the Pima Indians Diabetes Dataset from the National Institute of Diabetes and Digestive and Kidney Diseases [4]. The 768 patients are Indian females who are at least 21 years old [4]. The dataset contains the outcome and several prediction variables such as the number of pregnancies, insulin level, age, and so on [4]. Each sample with eight prediction variables and the outcome was taken into a Perceptron model for prediction. Perceptron is a binary classifier based on the linear decision function [1]. Due to the simplicity of the model, Perceptron is becoming popular [1].

2. Methodology

The methodology consists of three parts, data pre-processing, supervised learning, and limitations and improvements.

2.1. Data Pre-processing

The Pima Indian dataset used is scaled to the same order between -1 to 1 since some of the features in the dataset have a large difference in their range of values. For example, diabetes pedigree function ranges from 0 to single digit, while the 2-Hour serum insulin (mu U/ml) is in three digits. The algorithm reaches convergence faster if the features have the same order of range [2]. The scaled dataset is separated into training, validation, and testing data in a ratio of 60%, 20% and 20% respectively. The model is trained on training data. Then, it is evaluated on validation data with different parameters, such as different initial weights and learning rates in the experiment section. The hypothesis with the parameters that give the highest validation accuracy will be further evaluated on the testing data.

2.2. Supervised Learning

The Perceptron model consists of a linear function, an activation function, and a cost function. The linear function takes each feature from each sample as input, multiplies them to a corresponding weight and adds them up to calculate the weighted sum.

$$linear\ output = \sum_{i=1}^{n} x_i * w_i + b \tag{1}$$

n is the number of features
x_i is the ith feature of a sample
w_i is the corresponding weight for the ith feature
b is the bias term

the bias term can be set as w_0 and x_0 can be set as 1 for simpler expression.

$$linear\ output = \sum_{i=0}^{n} x_i * w_i \tag{2}$$

which is equivalent to the inner product of feature vector and weight vector

$$linear\ output = \langle X, W \rangle \tag{3}$$

The linear output is then passed to an activation function which is a sign function in our model. The output is a binary number, either 1 or -1, indicating the positive and negative diabetes predicted results respectively.

$$y* = sign(\langle X, W \rangle) \tag{4}$$

$y*$ is the predicted result

Each of the predicted results is compared to the label. The cost function represents the sum of errors for each misclassified sample.

$$C = \frac{1}{m}\sum_{j=1}^{m} max(0, -y_j\langle X_j, W\rangle) \tag{5}$$

C is the cost function
j is the number of samples
X_j is the feature vector for the jth sample
y_j is the label for the jth sample
m is the total number of samples

Gradient descent is used to minimize the cost function and update the weight. The derivative of cost function with respect to weight is as followed:

$$\frac{\partial C}{\partial w} = -\frac{1}{m}\sum_{j=1}^{m}(-y_j x_j \delta) \tag{6}$$

δ = 1 for misclassification
δ = 0 for correct classification

The weight is updated in each iteration with a specific learning rate.

$$W_{t+1} = W_t + \eta\frac{1}{m}\sum_{j=1}^{m}(-y_j x_j \delta) \tag{7}$$

η is the learning rate
t is the number of iterations

The algorithm continues until all the iterations are executed. The resulting weight is used for the prediction.

2.3. Limitations and Improvements

A single-layer Perceptron model with one neuron can only work with linearly separable data. If the data is not linearly separable, the accuracy cannot reach 100%. The majority of real-life data are not perfectly linearly separable. This might decrease the accuracy of the model and cause fluctuation in accuracy during learning. Nevertheless, we can improve the model accuracy by increasing the number of neurons and layers (Multi-Layer Perceptron). The working principle of Multi-Layer Perceptron is similar to Single Layer Perceptron. It consists of the input layer, the hidden layer and the output layer. The input layer takes in the input data and the output layer produces the predicted output. Each neuron in the hidden layer has a non-linear activation function. The combination of non-linear activation functions creates a non-linear decision boundary. The cost can be minimized by gradient descent and backpropagation. Backpropagation means calculating the gradient of the cost function with respect to each weight using the chain rule. It starts with the last layer and goes backwards. The resulting non-linear decision boundary can give a higher accuracy to real-life data.

3. Experiments and analysis

The experiments and analysis section consists of three parts, verification, general analysis and the experiments.

3.1. Verification

We use an artificially generated linearly separable dataset to verify our Perceptron model. The dataset has 1000 samples, each with two features and a label, classified into two classes. We use a random initial weight between 0 to 0.1 and a learning rate of 0.001 to run the model for 10 iterations. As shown in figure 1, both the training and validation accuracy reach 100%. The testing accuracy is also 100%. The Perceptron model successfully classifies the linearly separable dataset.

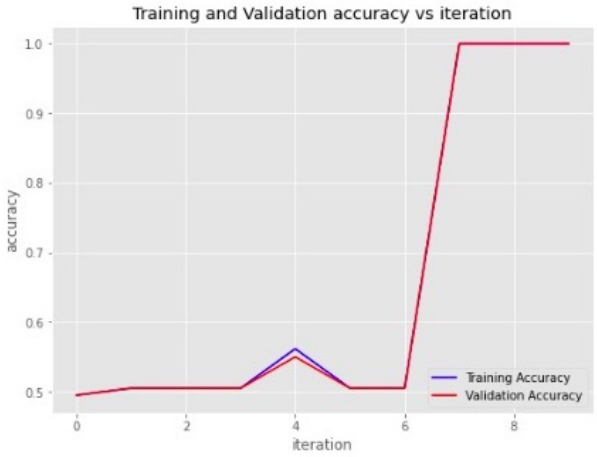

Figure 1. The accuracy versus iteration on the artificially generated linearly separable dataset. The accuracy converges to 100% in 10 iterations.

3.2. General Analysis

We demonstrate our model using the Pima Indian Diabetes dataset with a random initial weight between 0 to 1 and a learning rate of 1×10^{-4} for 700 iterations. As shown in Figure 2, the training and validation accuracy have similar results. Both accuracy increase with the number of iterations and converge to a range between 60% and 70%. The maximum validation accuracy 74.6% and the corresponding training accuracy is 69.3%. The accuracy fluctuation might be due to the non-linearity of the dataset. Since the dataset

may not be perfectly linearly separable, an update in weight for the correct classification of one data point might result in the misclassification of another data point. Therefore, the accuracy fluctuates within a range instead of converging to a specific percentage.

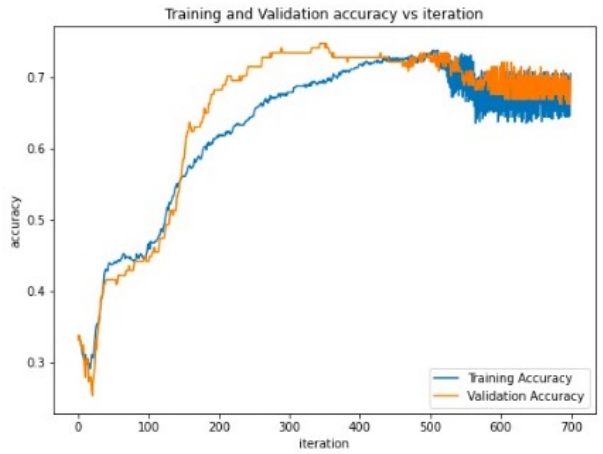

Figure 2. The training and validation accuracy versus iteration on Pima Indian Diabetes dataset with random initial weight between 0 to 1 and learning rate of 1×10^{-4} for 700 iterations. Both training and validation accuracy show a stable convergence after 500 iterations.

3.3. Experiment 1

In this experiment, we use initial weights from four different ranges. The experiment aims to find the best range of initial weights among these four ranges, which maximizes the accuracy and minimizes the number of iterations for convergence. Table 1 shows the maximum validation accuracy and the corresponding training accuracy for these four ranges of random initial weight. Figures 3,4,5, and 6 show the training and validation accuracy versus the iteration for these four ranges of random initial weight. As illustrated in Table 1, the random initial weight between 0 to 0.1 has the highest maximum validation accuracy (78.6%) and corresponding training accuracy (77.0%). It also converges much faster than other initial weights, as shown in figures 3,4,5 and 6. As the random initial weight range increases, the training and validation accuracy decreases and more iterations are required to reach convergence. For random initial weight between 5 to 6, the maximum validation accuracy and the corresponding training accuracy decreases to 46.1% and 42.2% respectively, and the result does not converge in 700 iterations. Therefore, initializing the weight with random numbers near zeros for the Perceptron model increases the accuracy and reduces the number of iterations for convergence. This might imply that the set of weights that gives the lowest loss is close to zero.

Experiment 1 (Random initial weight from different range)		
Initial Weight	Maximum Validation Accuracy (%)	Corresponding Training Accuracy (%)
Random between 0 to 0.1	78.6	77.0
Random between 0 to 1	74.7	69.3
Random between 1 to 2	73.4	66.3
Random between 5 to 6	42.2	46.1

Table 1. The maximum validation accuracy and corresponding training accuracy with random initial weight from four different ranges, learning rate = 1×10^{-4} and 700 iterations. As the random initial weight range increases, both the training and validation accuracy decreases.

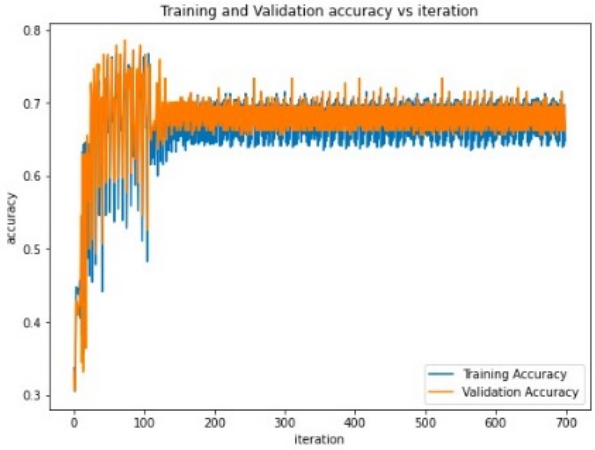

Figure 3. The training and validation accuracy versus iteration with random initial weight between 0 to 0.1, learning rate = 1×10^{-4} and 700 iterations.

3.4. Experiment 2

In this experiment, we study three different learning rates ($0.1, 1\times10^{-4}, 1\times10^{-6}$) on the Perceptron model. The best random initial weight range (0 to 0.1) from the previous experiment is used for this experiment. The experiment aims to find the best learning rate among these three learning rates, which increases the accuracy and reduces the number of iterations for convergence. Table 2 shows the maximum validation accuracy and the corresponding training accuracy among these three learning rates. Figures 7, 8, and 9 show the training and validation accuracy versus the iteration for these three learning rates. As illustrated in Table 2, the maximum validation accuracy for learning rate 0.01 and 1×10^{-4} are similar (77.9% and 78.6% respectively). However, in figures 7 and 8, for the learning rate of 0.01, the accuracy fluctuates much more than that of 1×10^{-4}, and

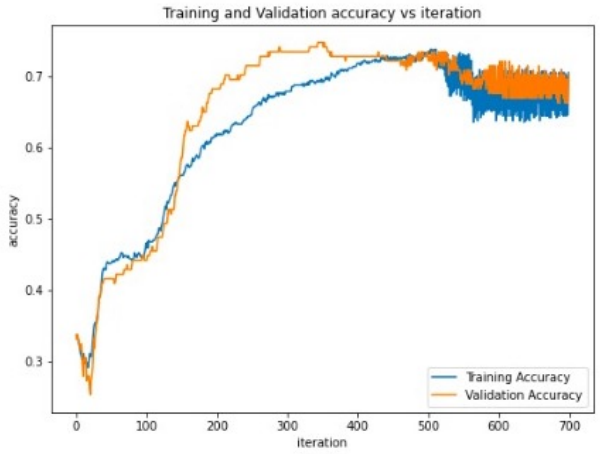

Figure 4. The training and validation accuracy versus iteration with random initial weight between 0 to 1, learning rate = 1×10^{-4} and 700 iterations.

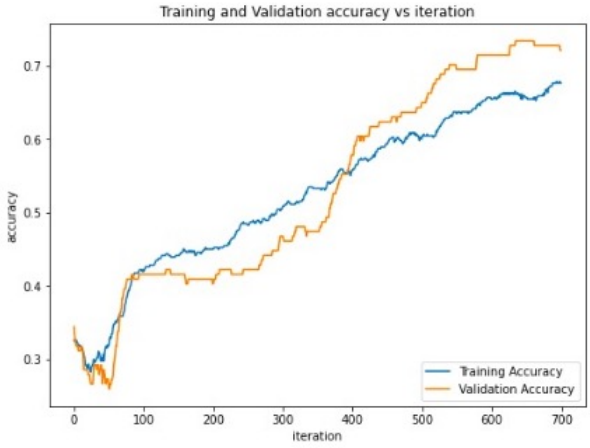

Figure 5. The training and validation accuracy versus iteration with random initial weight between 1 to 2, learning rate = 1×10^{-4} and 700 iterations.

it does not converge. It might be due to the overshooting of the minima in gradient descent. The result shows that the learning rate of 0.01 is too large for our Perceptron model. For the learning rate of 1×10^{-4}, the training and validation accuracy are the highest and converges within 60% to 70% in figure 8. The training and validation accuracy dropped significantly with the learning rate of 1×10^{-6} as shown in Table 2 and figure 9. It might take much more iterations for the loss to reach the minima in gradient descent. The result shows that the learning rate is too low. Therefore, 1×10^{-4} is the best learning rate among the three, which results in a good learning speed without jumping over the minima in gradient descent.

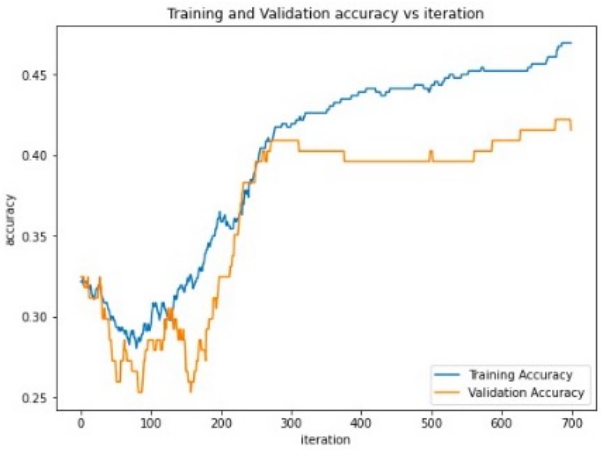

Figure 6. The training and validation accuracy versus iteration with random initial weight between 5 to 6, learning rate = 1×10^{-4} and 700 iterations.

Experiment 2 (Different learning rate)		
Learning rate	Maximum Validation Accuracy (%)	Corresponding Training Accuracy (%)
0.01	77.9	74.3
1×10^{-4}	78.6	76.9
1×10^{-6}	42.9	44.6

Table 2. The maximum validation accuracy and corresponding training accuracy with three different learning rates, random initial weight between 0 to 0.1 and 700 iterations.

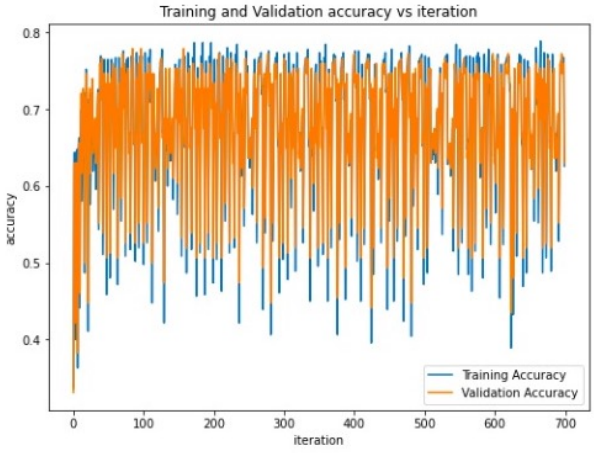

Figure 7. The training and validation accuracy versus iteration with learning rate = 0.01, random initial weight between 0 to 0.1, and 700 iterations.

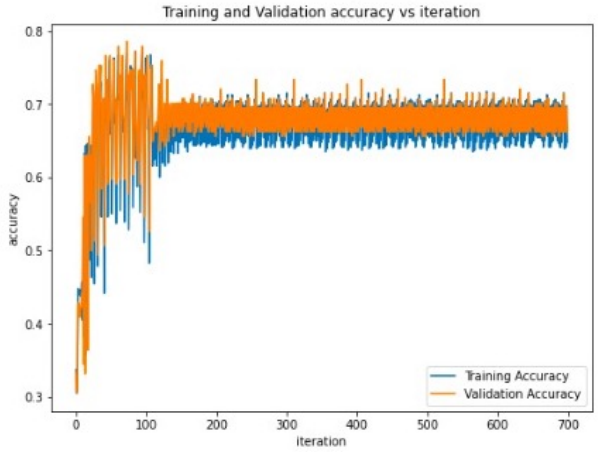

Figure 8. The training and validation accuracy versus iteration with learning rate = 1×10^{-4}, random initial weight between 0 to 0.1, and 700 iterations.

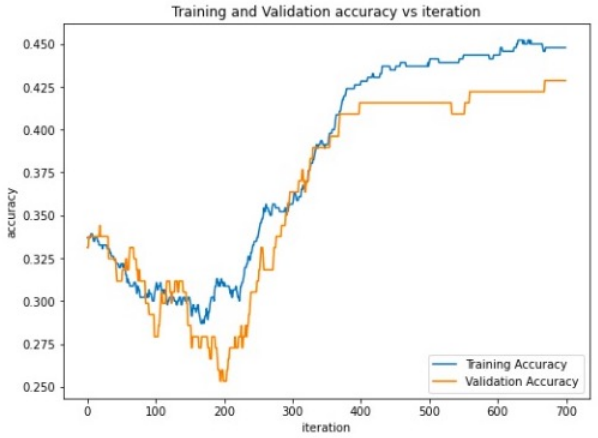

Figure 9. The training and validation accuracy versus iteration with learning rate = 1×10^{-6}, random initial weight between 0 to 0.1, and 700 iterations.

3.5. Experiment 3

In this experiment, we analyze the best random initial weight (between 0 to 0.1) and learning rate (1×10^{-4}) from the previous two experiments without bias. The purpose of this experiment is to examine the effect of bias on the performance of the model. The maximum validation accuracy for the weight with bias and without bias are similar (78.6% and 77.3% respectively). However, as shown in figures 10 and 11, the weight without bias results in a relatively large fluctuation in accuracy and does not converge. In contrast, the weight with bias shows convergence after around 100 iterations. Initializing the weight without bias might result in an additional restriction on the decision boundary as the decision boundary must pass through the origin. Hence, the model has a larger change in accuracy when updating the weight in each iteration, resulting in a greater accuracy fluctuation.

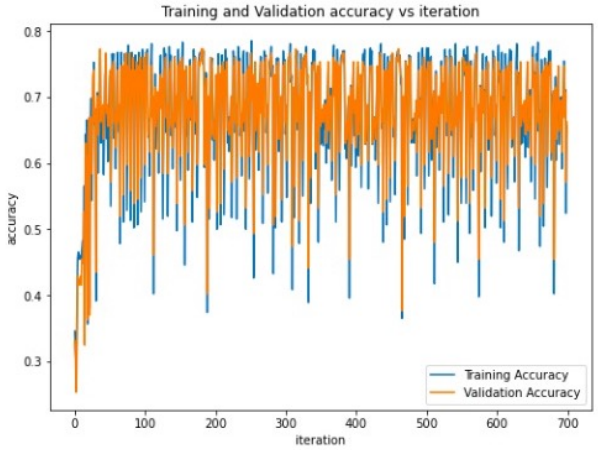

Figure 10. The training and validation accuracy versus iteration with learning rate = 1×10^{-4}, random initial weight between 0 to 0.1, and 700 iterations without bias

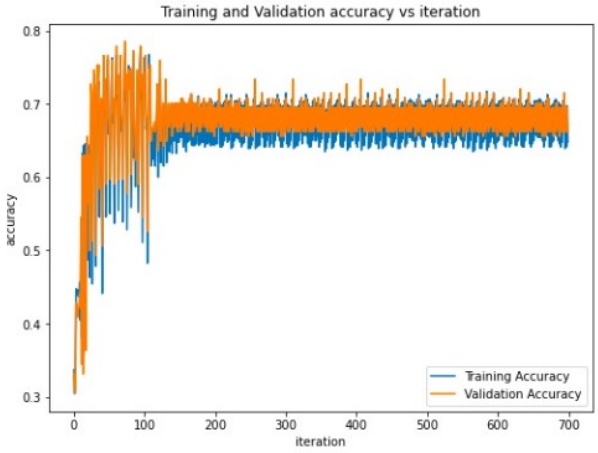

Figure 11. The training and validation accuracy versus iteration with learning rate = 1×10^{-4}, random initial weight between 0 to 0.1, and 700 iterations with bias

3.6. Testing Accuracy

After all experiments, we choose the best random initial weight (between 0 to 0.1) with bias and the best learning rate (1×10^{-4}) to be the parameters that result in the best validation accuracy. We apply the corresponding weight to the testing data. The testing accuracy is 82.5%.

4. Code

https://github.com/ChanKaHing/PerceptronAlgorithmforDiabetesPrediction.git

5. Conclusion

This paper presents a single-layer one-neuron Perceptron model for diabetes prediction. After the experiment and analysis, there are three main findings. First, a random initial weight range near zero might be a good choice for a Perceptron model in terms of accuracy and convergence. Second, there exists a range of optimum learning rates which provides a good learning speed without causing divergence in accuracy. Third, adding bias to the model increases the flexibility of the decision boundary and reduces the accuracy fluctuation. Future work may include improving the accuracy by using a Multi-Layer Perceptron model. It allows the model to combine outputs from two or more activation functions, resulting in a non-linear decision boundary. A non-linear decision boundary can separate real-life data better than a linear decision boundary, which gives higher accuracy.

References

[1] Murty, MN Raghava, R 2016, Support Vector Machines and Perceptrons: Learning, Optimization, Classification, and Application to Social Networks, Springer International Publishing AG, Cham.

[2] Rebala, G, Ravi, A Churiwala, S 2019, An Introduction to Machine Learning, Springer International Publishing AG, Cham.

[3] World Health Organization 2022, Diabetes, World Health Organization, n.d. <https://www.who.int/health-topics/diabetes/diabetestab=tab_1>

[4] The University of California Irvine Machine Learning Repository, Pima Indians Diabetes Database, n.d. <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>