# Perceptron

Sam Gilbert
a1737770

## Abstract

*The perceptron algorithm is one of the simplest machine learning algorithms. It is a form of supervised learning that*

## 1. Introduction

Perceptron is one of the simplest machine learning algorithms. Perceptron is a supervised learning algorithm that takes a combination of parameters to make a binary classification. Perceptron works by iteratively calculating a set of weights based on the dimension size of the features in the dataset, $D$. The output of which is a line or $D$ dimension plane that separates the observations into two categories. Perceptron calculates the updated weights iteratively and in an ideal world will converge to a $D$ dimension vector of values that separates all values into their correct groups. Where the data is not able to be separated cleanly into two groups the Perceptron algorithm will never converge. This requires a max iteration value to be set such that the problem is bounded. If a max iteration value is not set the Perceptron algorithm can keep moving back and forth trying to cleanly separate the training observations into the correct groups. Determining the optimal max iterations for the dataset will be investigated in the method section of the report.

In this project the Perceptron algorithm was implemented on the Pima Indians Diabetes Database open source dataset [2]. This dataset contains a set of eight medical predictor variables, the number of pregnancies the patient has had, their BMI, insulin level, plasma glucose concentration, blood pressure, skin fold thickness, age and their diabetes pedigree. The dataset also contains an outcome variable which is either the patient has diabetes or they don't have diabetes. The predictor variables are each one of the $D$ dimensions.

## 2. Method

As outlined in the introduction section the goal of the Perceptron algorithm is to separate the entries in the dataset into one of two binary groups. The data is read into a python array of dictionaries with the features being stored in a $D$ dimensional array and the outcome stored as a signed integer. The data is then split into a training and testing set using the scikit learn train test function. An 80-20 training testing split was used [1].

The Perceptron algorithm takes an input layer which is the set of features in each observation, let the vector of features for observation $i$ be defined as $x_i$. Let the collection of these observations be defined as $X$. Each of these observations have an outcome variable which is whether the patient has diabetes or not. Let the outcome for the $i_{th}$ observation be defined as $y_i$. This outcome variable is either $+1$ if the patient has diabetes or $-1$ if the patient doesn't have diabetes. Let the collection of all the outcomes for each observation be defined as $Y$. The Perceptron algorithm starts by creating a $D$ dimension vector of weights, defined as $W$. For this implementation the initial $W$ is set as a $D$ dimension zero vector. A bias value is also required which is defined as $b$ initially set to 0. This bias shifts the intercept of the dimensions linear separator away from 0 and is required as it may not be possible to split the data points if this doesn't occur.

The Perceptron algorithm works by calculating the dot product of $x_i$ and $W$ and adding the bias, $b$. The sign function is then passed the result and this is our activation function corresponding to the possible values of the outcome variable mentioned above. The sign function is defined as

$$sign(x) = \begin{cases} -1 & \text{if x} < 0 \\ 0 & \text{if x = 0} \\ +1 & \text{if x} > 0 \end{cases} \quad (1)$$

To determine if the perceptron has correctly classified the observation, the result of the sign function corresponds to the outcome variable possibilities of whether a patient has diabetes or not. Therefore, the whole process of making a prediction on an observation is defined as.

$$f(x) = sign((x_i \cdot W) + b) \quad (2)$$

Using the predictions from equation (2) the weight vector, $W$ can be updated during the training process. By multiplying the prediction by the actual class of the observation and checking the resulting value is $> 0$ the update logic can

be simplified. If the result is $> 0$ then the perceptron has made a correct prediction and does not need to be updated. This can be seen in the logic table below showing if a correct prediction is made the result will always be $> 0$ and if an incorrect prediction is made the result will be $<= 0$

| Truth | Prediction | Output |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Table 1. Prediction vs truth logic table for predictions

If the perceptron makes an incorrect prediction during the training phase $W$ is updated by multiplying $y_i$ with $x_i$. Let $W$ at the current time be defined as $W_t$. This update step is defined as

$$W_{t+1} = W_t + (y_i * x_i) \tag{3}$$

Similarly, $b$ needs to be updated in the case of an incorrect prediction. Let $b$ at the current time be defined as $b_t$. The update step is defined as

$$b_{t+1} = b_t + y_i \tag{4}$$

This process is repeated for every observation in the training set. In an ideal scenario this process would be repeated until every element has been correctly classified, however this is most often not possible. Where the data is not linearly separable on a given dimension the Perceptron algorithm will not be able to converge on a solution. To address this a max iteration value must be set which will set a stop point for the algorithm if it fails to converge. This max iteration value is the number of times the algorithm will iterate over the entire training dataset during the training phase. Different values will be investigated in the results section to examine the impact it has on the overall model performance.

## 3. Results

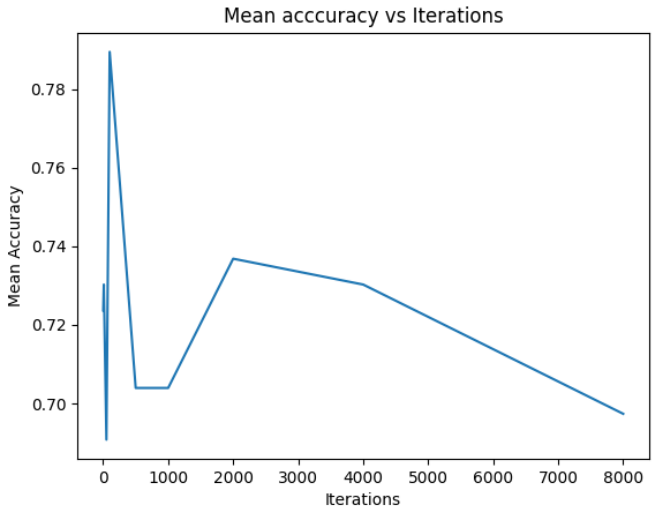
### 3.1. Max Iteration Investigation

As discussed above the max iteration value used during the training phase can have a big impact on the overall model performance. Allowing the model time to converge on an optimal or near optimal solution is paramount. To determine the appropriate magnitude of the max iteration value, various values were set during the training phase. A set seed was passed to the function that splits the data into the training and testing set to ensure each value of max iterations was being tested on the same data. To reduce the variance in the results, which will be investigated later, each max iteration value was run 10 times and the mean of the model accuracy was calculated.

| Iterations | Mean Accuracy |
|---|---|
| 1 | 0.723 |
| 10 | 0.730 |
| 50 | 0.753 |
| 100 | 0.789 |
| 500 | 0.703 |
| 1000 | 0.703 |
| 2000 | 0.736 |
| 4000 | 0.730 |
| 8000 | 0.697 |

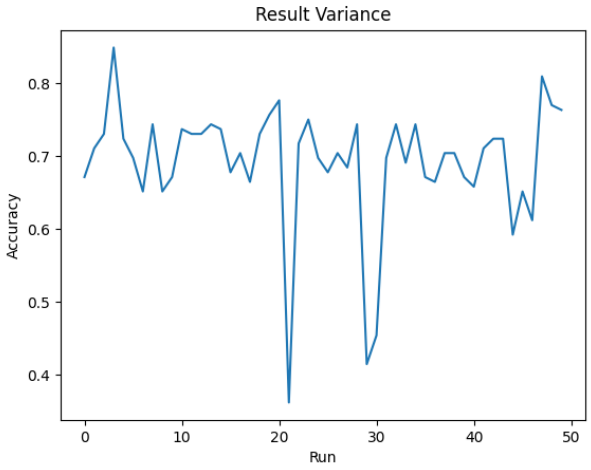
Table 2. Iterations vs mean accuracy

Plotting the results from the table above it can be observed that as the number of max iterations increases, the corresponding mean accuracy of the Perceptron algorithm also increases before reaching an equilibrium. For the highest magnitude max iterations the accuracy begins to drop off. This will be investigated in the discussion section. There is also a spike at the iteration value of 100 and this will be investigated in the discussion section.




### 3.2. Result Variance Investigation

The variance of the final model accuracy result is highly dependent on the data selected for the training set. Since the data is randomly selected, the final model $W$ vector and $b$ can vary greatly. The impact of this was investigated by running the Perceptron algorithm 50 times with an iteration value of 2000 which was found to provide the best, stable results.




From the plot above it can be observed that the variance of the Perceptron algorithm can vary greatly depending on the initial split. This can possibly be mitigated and approaches for this will be investigated in the discussion section

## 4. Discussion

The Perceptron algorithm can be very effective at predicting diabetes using the Pima Indian dataset. Depending on the number of iterations and the data chosen for the training set accuracy as high as approximately 0.83 has been observed. It can be highly variable in its performance due to the datasets small size and the lack of a solution that splits all the observations into the correct groups. With the dataset only containing 768 observations, when it is split into the training and testing sets the training set only contains 614 observations. This small set of observations leads to variability in the models performance depending on the sample selected. To address this a bigger dataset could be used which would result in a reduced impact on the final results caused by the small number of observations.

For this report, outlier data was not accounted for. This could have compounded with the small dataset size to increase the variability of the final results. If the training dataset contained a high proportion of observations with outlier data the final $W$ would not result in optimal results when run against the testing set. Future work for this problem would be to address the outlier data through omitting observations that contain outlier values or devising a method of keeping them out of the training set which would likely increase the models average performance.

## References

[1] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation.

[2] Kaggle. Pima indians diabetes database — kaggle.com. https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database. [Accessed 10-09-2024].