Perceptron for Predicting Diabetes

Christopher Hamilton
The University of Adelaide
christopher.hamilton@student.adelaide.edu.au

Abstract

This paper presents an experimental analysis of using a perceptron to predict if an individual has diabetes based on a set of variables. As the prevalence of diabetes in Australians is increasing slowly, going from a rate of 3.3% in 2001 to a rate of 5.3% in 2022 [4], predicting diabetes without the need for a specific blood test is an important problem to solve. As a chronic condition, which can lead to "health complications, such as heart attack, stroke, and limb amputation"[4], diabetes is a disease that needs to be managed in individuals that have it. As of 2014, with 8.5% of adults having diabetes, and more recent studies determining that diabetes is a direct cause of 1.5 million deaths[8], it is more important than ever to accurately diagnose the disease. The objective of the analysis done here is to determine if a perceptron can be used to accurately predict whether an individual has diabetes based on a set of medical variables. A perceptron works by producing a weighted sum of the input variables provided to it, and calculating the prediction based on the weights.[3] In the training phase, the weights are updated based on the input variables used, when the data is classified wrong. This is repeated for a number of iterations, and a learning rate has been applied to update the weights by a smaller factor. The perceptron was trained on 80% of the diabetes data provided, and then tested on the remaining 20%. Multiple experiments were run, using different training parameters, and an accuracy of over 71% was achieved with a learning rate of 0.01 and 100 iterations. More research could be done using different machine learning algorithms to achieve a higher accuracy of prediction.

1. Introduction

The prediction of diabetes in individuals without the need for specialised testing may be beneficial for the early diagnoses of the disease as well as increasing the number of people who get tested. Given a set of data relating to a person's health, we may be able to accurately identify whether or not that individual has diabetes, without the need for a specific blood test. The dataset used for the analysis in this paper can be found at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html` under the heading 'diabetes'. This data was processed from the dataset available at `https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database`. The set of variables available in the dataset includes the number of pregnancies, glucose concentration in an oral glucose concentration test, blood pressure, skin thickness, insulin level, BMI, diabetes pedigree function and age. These independent variables act as predictor variables for the single dependent, outcome variable in the dataset, whether or not the individual has diabetes. As the prediction is either the individual having diabetes or not, the problem to be solved is a binary classification problem.

A single perceptron is able to classify data into two classes through linear separation of the data. The classification is done through the use of a weighted sum that has a threshold applied and depending on if the sum is over or under the threshold, the data is classified into either of the classes.

2. Method

The perceptron is an adaptive system that is able to recognise patterns in data as well as model the functions of the brain. It was first implemented in 1957 by Frank Rosenblatt at Cornell Aeronautical Laboratory as a way to classify inputs without the need for human action, similar to the human brain [5]. Through research into the physical nervous system and the human brain, discoveries have been made into neuron pathways, and how neurons interact with one another. In humans, it has been discovered that the initial network of connections between neurons is mostly random and as neural activity occurs during development, the reactions of cells to a stimulus changes [1]. Another key result that was found is that when a many stimuli are applied, connections are often formed between the same sets of neurons for similar stimuli and are formed between different sets of neurons for stimuli that are unalike. [**?**]

These discoveries have been applied in the development

of the perceptron for the binary classification of data. The result of the similar stimuli causing activation for similar neurons has been taken and from this, the perceptron has been designed to classify data to the same class that similar training data was labelled as. A perceptron takes in an input of multiple variables, and based on a set of weights that it has been trained for, generates a weighted sum of these variables. For a perceptron, this weighted sum is used to classify the output into either of the two possible classes, if the sum is greater than or equal to 0, the will be given the positive class (+1) and if it is less than 0, it will be given the negative class (-1).[3]

In order for the perceptron to be able to accurately classify data, it must have weights trained based on a set of inputs which are labelled with correct classes already. The perceptron training algorithm involves setting all weights incorrectly at first, and then iteratively updating the weights based on the product of the training class and the training input data. This algorithm can be written mathematically as:

Assume:

$$g(\vec{x}; \vec{w}) = sign(\vec{x} \cdot \vec{w})$$

Where $\vec{x}, \vec{w} \in \mathbb{R}^d$, and $y \in \{-1, 1\}$

For a set of training data: $\{(\vec{x_i}, y_i)\}_{i=1}^n$,

a step size: η,

and a number of iterations T

The perceptron training algorithm is as follows:

$\vec{w} \leftarrow \vec{0}$

for $t = 1$ to T **do**

$$\vec{w_{t+1}} = \vec{w_t} + \eta \sum_{i=1}^{n} (y_i \vec{x_i} 1_{\{y_i(\vec{x_i} \cdot \vec{w_t}) \leq 0\}})$$

end for

[6]

We denote the trained weights of the perceptron as w^* and this is the same as w_T, the weights as updated at the end of the final epoch.

Prediction using this trained perceptron is then done by taking the dot product between the input vector and the weights, to produce a predicted class label, $y^* \in \{-1, 1\}$.

$$g(\vec{x}; \vec{w^*}) = y^* = sign(\vec{x} \cdot \vec{w^*})$$

Since a weighted sum is used as the input to the perceptron, computers are able to very quickly perform the dot products and summations required for the training of a perceptron, as well as the classification of data. This makes the method very useful to quickly classify linearly separable data into one of two classes, given that the weights have been appropriately set to separate the data for the problem to be solved. Setting the weights correctly is the task of the training algorithm that has been outlined above and means that the set of training data points should be large to allow for good generalisation to data points that have not been seen.

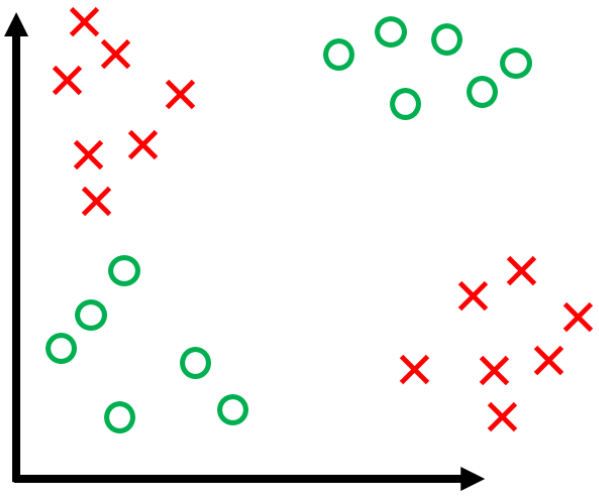

Figure 1. Example of non-linearly separable data [7]

The perceptron while able to converge to a prediction for linearly separable data, has some limitations. The first limitation of the perceptron algorithm is that it is only able to converge if the data is linearly separable. [7] This limits the use cases that the perceptron has, a simple example of this is shown in Figure 1. Since a lot of data in the real world will not be perfectly linearly separable, the perceptron trained and tested on real data is not likely to have 100% accuracy. However, this may be acceptable in cases where the perceptron is able to accurately classify test data, even though convergence was not achieved. An example of this is shown in Figure 2. The data contains two classes, but these cannot be linearly separated, any line drawn to separate the points in this two dimensional space will result in some of the data points being classified incorrectly. In the case of Figure 1, the perceptron is not a good solution to classify data points into either of the available classes, as the accuracy will be low. However, in the case of Figure 2, even though the data is not linearly separable, the perceptron is still an effective method for classifying data points, in the example, only three points are incorrectly labelled based on the linear separation provided.

Another limitation of the perceptron is that it is only able to perform binary classification, and this results in only two options for the predicted class. This works well in data sets where the outcome variable is some sort of indicator variable, however for other use cases, such as data that needs to be classified into three classes, it will not work.

3. Experimental Analysis

In order to achieve the aim of the experiment, to use a perceptron to predict diabetes, given a set of medical

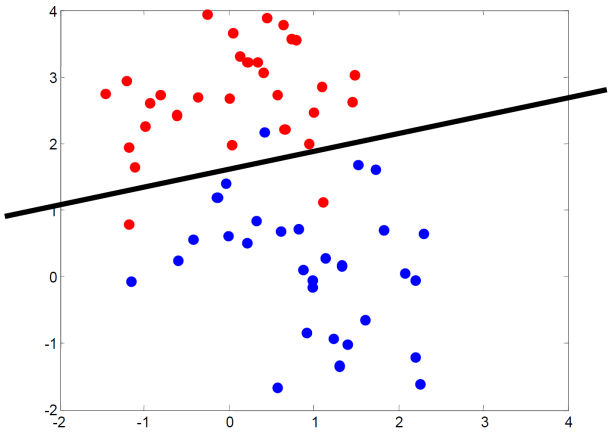

Figure 2. Classification of non-linearly separable data [2]

variables, the perceptron algorithm described above in the method section was implemented using Python. The perceptron needs to have its weights trained in order to produce the separation of the data, and accurately perform predictions. The diabetes data has been processed, with the features being scaled, and the outcome being set to -1 or 1, as required by the perceptron. The diabetes data used can be found at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`, under the name 'diabetes_scale'. For the purposes of training the perceptron and testing its accuracy, the diabetes data was randomly split into two sets, one for training and one for testing, with 80% of data used for training the perceptron and 20% used for testing.

The parameters that can be applied to the training algorithm above are the number of iterations, and the learning rate. In order to determine the optimal learning date and number of iterations, the perceptron should be trained with each set of parameters, and the accuracy should be compared. The values chosen for this are training with 10, 100 and 1000 iterations, as well as 0.1, 0.01 and 0.001 learning rates. The perceptron was run 10 times for each of these sets of values, and the average accuracy produced from predicting the class of test data has been calculated.

After running the code multiple times, it can be observed that the accuracy produced can be quite varied between a run. This implies that the data is likely not linearly separable, and since the weights are not able to converge, not all data points can be classified correctly. However, taking an average may be able to give a better representation of the accuracy for a given learning rate and number of iterations.

Based on the results seen in Table 1, Table 2, and Table 3, the highest accuracy was achieved using a learning rate of eta = 0.01 and training for 100 iterations. For the experiments where training was only done for 10 iterations, the accuracy on average was lower for all learning rates, how-

Run	Accuracy (%) $\eta = 0.1$	Accuracy (%) $\eta = 0.01$	Accuracy (%) $\eta = 0.001$
1	33.33	54.90	52.29
2	38.67	70.59	39.87
3	69.93	62.09	58.17
4	66.01	56.21	51.63
5	73.86	66.01	66.01
6	71.24	40.52	69.28
7	64.71	73.86	66.01
8	75.16	69.93	34.64
9	74.51	50.98	71.90
10	69.28	65.36	35.95
Ave	63.66	61.05	54.58

Table 1. Results of running experiments with 10 iterations

Run	Accuracy (%) $\eta = 0.1$	Accuracy (%) $\eta = 0.01$	Accuracy (%) $\eta = 0.001$
1	66.01	75.16	75.82
2	69.93	80.39	75.82
3	66.67	66.67	71.90
4	67.97	77.78	64.71
5	58.82	55.56	72.55
6	56.21	73.20	60.13
7	75.16	70.59	79.08
8	62.75	66.01	70.59
9	78.43	72.55	69.28
10	67.97	75.16	68.63
Ave	66.99	71.31	70.85

Table 2. Results of running experiments with 100 iterations

Run	Accuracy (%) $\eta = 0.1$	Accuracy (%) $\eta = 0.01$	Accuracy (%) $\eta = 0.001$
1	69.28	67.97	71.90
2	63.40	71.90	64.05
3	64.71	64.71	71.24
4	58.82	77.12	64.71
5	75.16	79.08	70.59
6	66.67	55.56	64.71
7	63.40	66.67	64.71
8	74.51	66.01	73.86
9	75.82	65.36	70.59
10	74.51	67.97	59.48
Ave	68.63	68.24	67.58

Table 3. Results of running experiments with 1000 iterations

ever, it can be noted that for less iterations, a higher learning rate was more effective in producing a higher accuracy model. For experiments done with 1000 iterations, it can be seen that the accuracy is very close to the highest accuracy produced, and that the learning rate has only a small effect on the average accuracy. This is likely because after this larger number of iterations, the model will reach similar weights, regardless of the learning rate used.

4. Code

The code written to implement the perceptron algorithm can be found at `https://github.com/CHamilton0/COMP-SCI-7318-Deep-Learning-Fundametals`, under the 'Assignment1' directory. It has been implemented in Python 3.11.9. To load the diabetes data, run the perceptron training algorithm and run predictions on the data, ensure numpy is installed with:

```
$ python3 -m pip install numpy
```

and then run the code with:

```
$ python3 perceptron.py
```

A helper script has been written in order to load the diabetes data in a form that the perceptron can work with. This loading function has been implemented in the 'load_diabetes.py' script. It will open the diabetes data file, and split the data in each line into the vector of variables and the outcome. These tuples of inputs and outcomes are then shuffled to randomise the data that is used for training and testing. In investigating the training and testing data, it was found that some rows in the data do not have all the variables set, for rows where not all variables are set, the script to load the data will leave the unset value as zero in the input vector, this will result in it not having an effect on the dot product done as part of training or classification.

The 'perceptron.py' script contains the implementation of the perceptron algorithm used. Weights are initialised to zero, and are updated for the training data that the perceptron has classified incorrectly. This is repeated for the number of iterations specified by T. Finally, the test data is classified with the trained perceptron, as described in the algorithm, and checked against the label. The accuracy, as a percentage of correct classifications on the test data, is calculated and displayed in the terminal.

5. Conclusion

The approach outline in this paper, using a perceptron to predict diabetes based on medical variables, has proven effective, achieving an accuracy of 71.31%. This approach to predicting diabetes does require knowledge of multiple medical variables in an individual, however, it does not require the specific blood test for the glucose. While the accuracy of this method is not high enough to be used as the sole method to determine if an individual has diabetes, this method could be used by medical professionals regularly, with the result of this being used to decide if a glucose test is required for the diagnosis. By doing this, medical professionals may be able to diagnose diabetes in more of the population, allowing people to manage their diabetes earlier, and reduce or prevent the effects of diabetes.

References

[1] H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34(1):123–135, January 1962.

[2] Kalev Kask and Alexander Ihler. Machine learning and data mining: Linear classification. `https://ics.uci.edu/~kkask/Spring-2018%20CS273P/slides/05-linClassify.pdf`.

[3] Xiaoyao Liang. Chapter 1 - theoretical basis. In Xiaoyao Liang, editor, *Ascend AI Processor Architecture and Programming*, pages 1–40. Elsevier, 2020. `https://www.sciencedirect.com/science/article/pii/B9780128234884000011`.

[4] Australian Bureau of Statistics. Diabetes, 2022. `https://www.abs.gov.au/statistics/health/health-conditions-and-risks/diabetes/latest-release`.

[5] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[6] Professor Javen Shi and Anthony Dick. Lecture 3 supervised learning: Knn, perceptron, logistic regression.

[7] Killian Weinberger. Lecture 3: The perceptron, 2017. `https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html`.

[8] WHO. Diabetes, April 2023. `https://www.who.int/news-room/fact-sheets/detail/diabetes`.