Predicting Diabetes with Perceptron

Samridhi Gupta
The University of Adelaide
samridhi.gupta@adelaide.edu.au

Abstract

Diabetes occurs when the body either produces insufficient insulin or is unable to use the insulin it generates effectively. This study aims to predict diabetes by applying the Perceptron algorithm to a dataset that includes data on Pima Indian women aged 21 and older. A simple Perceptron model is implemented from scratch, is trained on the dataset and its accuracy is evaluated.

1. Introduction

The dataset contains various medical predictor variables along with a target variable called "Outcome." The predictor variables include the number of pregnancies a patient has had, their BMI, insulin levels, age, and others. The dataset comprises 768 rows and 9 columns, with the final column representing the outcome (0 or 1), indicating whether the individual has diabetes or not. The Perceptron algorithm is a type of linear classifier used for binary classification tasks, which adjusts its weights based on the input data to minimize classification errors. It works by iteratively updating the weights during training until the model can separate the two classes with a linear decision boundary. The Perceptron, while effective for binary classification with linearly separable data, struggles when the data is not linearly separable, unlike more advanced algorithms like Support Vector Machines and Logistic Regression, which can handle non-linear relationships through kernel tricks or probabilistic interpretations [4]. Additionally, Multi-Layer Perceptrons and Neural Networks outperform the basic Perceptron by introducing non-linearity and deeper architectures, enabling them to capture complex patterns in data [4]. These advanced models are often more robust in real-world scenarios where data is rarely perfectly linearly separable.

2. Method description

The Perceptron algorithm works by taking input features and applying weights to them to compute a weighted sum. This sum is then passed through a step activation function, which classifies the output into one of two categories. During training, the algorithm adjusts the weights based on the errors between the predicted and actual outcomes. The process continues iteratively until the model finds an optimal set of weights that minimizes classification errors, enabling it to draw a linear decision boundary between the two classes [3]. If **X** represents a vector of input values, **w** a vector of weights, and **b** a bias vector, the weighted sum can be calculated as:

$$weighted\ sum = (X.w) + b$$

where $(x.\ w)$ is the dot product of the vectors **x** and **w**. This weighted sum is passed to an activation function, which generates an output of 0 or 1 based on whether the sum surpasses a certain threshold, in this case, 0. The output from this step function is the model's prediction.

During the learning phase, the algorithm compares its prediction to the actual value and adjusts the weights accordingly to achieve better predictions. The aim of this process is to determine the optimal weights (**w**) and bias (**b**) for accurate classification. This is done using the Perceptron's weight update rule, given as:

$$w_i = w_i + r(y_{true} - y_{pred})X_i$$

w_i is the ith element of the vector of weights (**w**)
r is the learning rate
y_{true} is the true value of y in the training set
y_{pred} is the predicted value of y by the model during training
X_i is the ith element of the input vector (**X**)

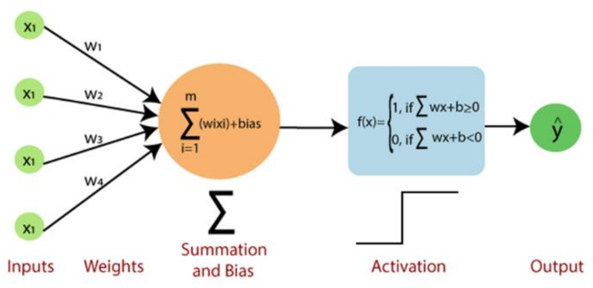

Fig 1. Structure of the Perceptron [3]

3. Experiments and Analysis

3.1 Exploratory data analysis

'Pregnancies' is a quantitative variable. So, a histogram and a boxplot is a good way to analyze it.

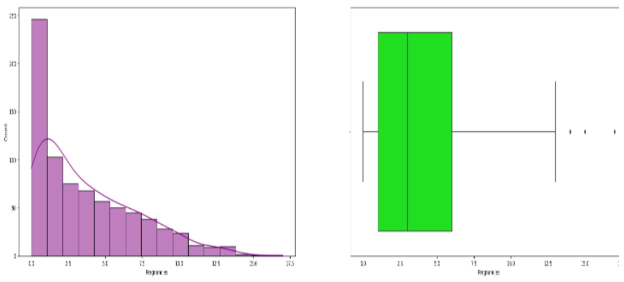
Fig 2. Univariate analysis of pregnancies

There are 3 outliers for pregnancies, they can be removed. I chose to not remove them because outliers can sometimes be useful. The median is 3.0 and the maximum is 17. The histogram is right-skewed.

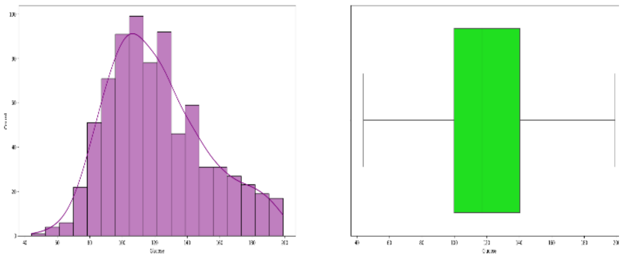
Fig 3. Univariate analysis of Glucose

We similarly analyze the rest of the features. ‘Glucose’ has a comparatively uniform distribution and no outliers.

The median of glucose is 117 and the maximum value is 199. The histogram of ‘SkinThickness’ is highly imbalanced and right-skewed. There are also many outliers.

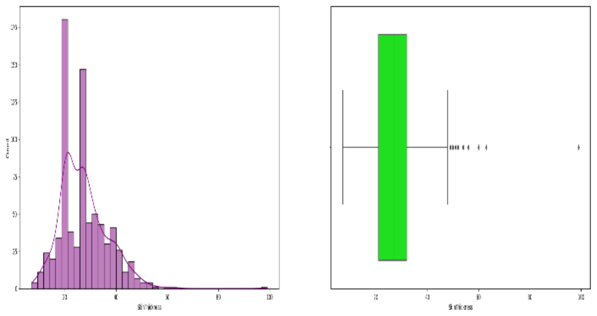
Fig 4. Univariate analysis of SkinThickness

By using similar methods, I obtained the histograms of all the numerical features. We see that the distribution of Insulin is highly right-skewed. The features ‘Age’ and ‘DiabetesPedigreeFunction’ are also akewed. So to conclude, the dataset is more or less imbalanced and there needs to be some sort of transformation needed.

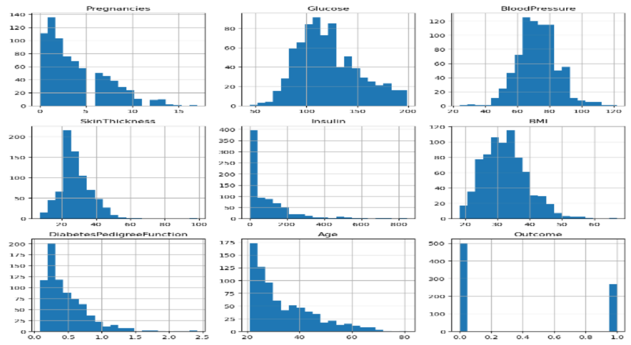

Fig 5. Histograms of all the features

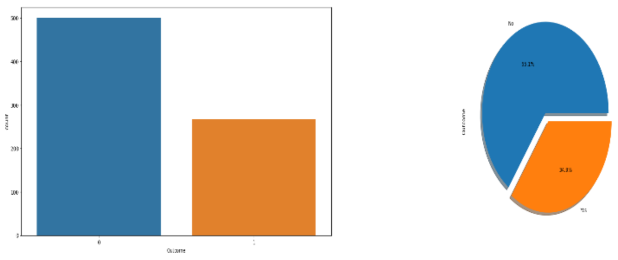
Fig 6. Pie Chart of ‘Outcome’ variable/target variable

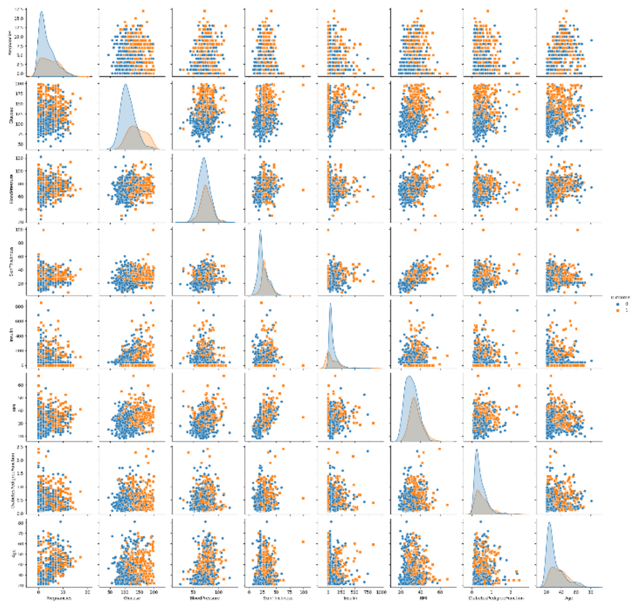
Fig 7. A pair plot of the features of the dataset

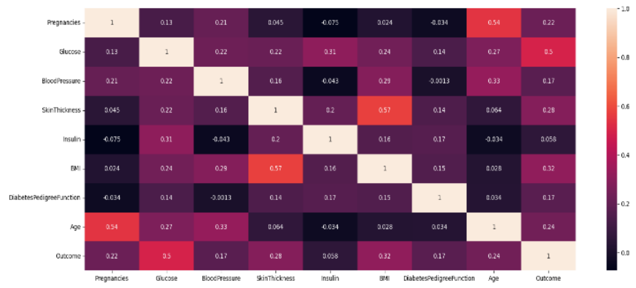
Fig 8. Correlation matrix of the features

The correlation matrix shows how strongly each feature is related to others and the target variable. Features with a high correlation to the target are more likely to be predictive [2].

3.2 Pre-processing

As I observed from the EDA that the dataset is having some highly imbalanced features, I felt the need to impute data using the median. Firstly, I checked for null values. There weren't any null values in the dataset. The Python function, `calc_median()`, performs median imputation on missing values in a dataset by calculating the median of a specific variable (`var`) grouped by a categorical target variable (`target`). Next, the `median_impute()` function is designed for efficient imputation of missing values in a dataset based on medians.

These two functions are then applied on all the numerical features to impute data with the median values. Imputing missing data using the median has the key advantage of being robust to outliers, as the median represents the middle value of a dataset and is not affected by extreme values like the mean. This makes it particularly useful when dealing with skewed distributions.

3.3 Implementing the algorithm

The Perceptron algorithm was implemented from scratch and applied to the Pima Indian Diabetes dataset for binary classification. It begins by splitting the dataset into training and testing subsets using the `train_test_split` function. The Perceptron class is initialized with a learning rate and a fixed number of iterations [1].

Learning rate	Accuracy score
0.3	0.72017
0.1	0.72017
0.01	0.72017

Table 1. No. of iterations are 5000

It can be observed from the above table that the accuracy scores do not vary by changing the learning rate. During training, the `fit` method iteratively adjusts the model's weights and bias. based on the input data using a simple step function to determine the output. The activation function used in this implementation is the step function given by:

$$f(x) = \begin{cases} 1, if\ x \geq 0 \\ 0, if\ x < 0 \end{cases}$$

The model updates its parameters by comparing the predicted and actual labels and adjusting them to minimize misclassifications. After training, the `predict` method is used to classify the test data, and the model's accuracy is evaluated by comparing predictions to actual labels using the `accuracy_score` function from `sklearn`.

No. of iterations	Accuracy
2000	0.70716
3000	0.70716
5000	0.72017
10000	0.68112

Table 2. The learning rate is 0.1

No. of iterations	Accuracy
2000	0.70716
5000	0.72017
10000	0.68112

Table 3. The learning rate is 0.01

The step function is used in the Perceptron because it aligns with the algorithm's primary goal of binary classification by providing a clear, hard decision boundary between two classes. The Perceptron outputs either 0 or 1, based on whether the weighted sum of inputs is above or below a threshold, making the step function a natural fit for this type of binary decision-making. Other activation functions, such as the sigmoid or ReLU, produce continuous outputs or add non-linearity, which are not necessary for the simple linear classification task the Perceptron is designed for. The step function is also computationally efficient, making it ideal for the Perceptron's straightforward learning process.

4. Method implementation

The code is available at https://github.com/samridhi20-hub/Deep-Learning_a1901641/blob/main/a1901641_Deep%20Learning_A1.ipynb

The code implements the Perceptron algorithm from scratch and applies it to the Pima Indian Diabetes dataset for binary classification. It begins by splitting the dataset into training and testing subsets using the `train_test_split` function. The Perceptron class is initialized with a learning rate and a fixed number of iterations. During training, the `fit` method iteratively adjusts the model's weights and bias based on the input data using a simple step function to determine the output. The model updates its parameters by comparing the predicted and actual labels and adjusting them to minimize misclassifications. After training, the `predict` method is used to classify the test data, and the model's accuracy is evaluated by comparing predictions to actual labels using the `accuracy_score` function from `sklearn`. The code thus demonstrates a custom implementation of a binary classifier that learns from input data to make accurate predictions [2].

5. Reflection on project

This experiment aimed to predict diabetes using a Perceptron algorithm implemented in Python. Different learning rates and iteration counts were tested, showing that accuracy improved as iterations increased, while the learning rate had a smaller effect. The model's performance could be enhanced by tuning other parameters, such as the activation function, or adding hidden layers. While the Perceptron is an effective linear classifier, it struggles with non-linearly separable data. This suggests that while the Perceptron is efficient for simple tasks, its limitations become apparent with more complex datasets. Enhancements such as incorporating multi-layer perceptrons or switching to non-linear classifiers like support vector machines or neural networks could address these issues. Additionally, further experimentation with feature engineering or regularization techniques might improve performance, making the model more robust in real-world applications where data is rarely linearly separable.

6. References

[1] Siddhardhan 2023, *Building Perceptron from scratch in Python | Deep Learning Course*, YouTube, 1 July, viewed 26 September 2024, <https://www.youtube.com/watch?v=JlXrqeqyKBo&list=PLfFghEzKVmjsdCvJeiNqL3e3djKmCCuBd&index=6>

[2] Sparsh Analytics 2021, *COMPLETE SOLUTION OF KAGGLE- PIMA INDIAN DIABETES DATASET 92% ACCURACY (1 of 3)*, YouTube, 29 March, viewed 26 September 2024, <https://www.youtube.com/watch?v=wrKSkt6MAwc>

[3] Kilic, I 2023, *Perceptron Model: The Foundation of Neural Networks*, Medium, viewed 26 September 2024, <https://medium.com/@ilyurek/perceptron-model-the-foundation-of-neural-networks-4db25b0148d>

[4] James, G, Witten, D, Hastie, T & Tibshirani, R 2013, *An Introduction to Statistical Learning with Applications in R*, New York Springer, < https://link.springer.com/book/10.1007/978-1-0716-1418-1>