Collabits Journal
Vol. 1 No. 3 September 2024 : 198-207
E-ISSN : 3046-6709, P-ISSN : 3062-8601
https://publikasi.mercubuana.ac.id/index.php/collabits

Neural Network Classification to Determine the Likelihood of Diabetes Using Python Programming Language

Rafif Syari Hidayah[1], Yudha Andika Istanto[2*]

[1,2] Informatics Engineering Study Program, Universitas Mercu Buana

*Coressponden Author: yudhaandikaistanto@gmail.com

Abstract - Diabetes is a global health problem that affects millions of people worldwide. Predicting a person's risk of developing diabetes can be an important first step in disease prevention and management. In this study, we propose the development of a predictive model for diabetes using Neural Network (NN) technique with implementation using Python. The data used in this study consists of clinical information that includes factors such as pregnancy, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, and age. The model development process involves data pre-processing, selection of relevant features, model training, and performance evaluation using appropriate metrics. The experimental results show that the developed NN model has a good ability in predicting diabetes risk. The main contribution of this research is the use of NN techniques and Python coding in the development of predictive models for diabetes, which can provide useful guidance for medical practitioners in supporting disease prevention and management efforts. Future studies can extend this research by considering additional factors and improving the accuracy of the model by using more complex approaches.

Keywords :
Diabetes;
Prediction;
Neural Network;
Python Coding;
Predictive Model;
Model development;
Data pre-processing;
Performance Evaluation;
Disease Prevention;
Disease Management;

Article History:
Received: 12-06-2024
Revised: 19-07-2024
Accepted: 28-08-2024

Article DOI : 10.22441/collabits.v1i3.27300

1. INTRODUCTION

Diabetes is a chronic disease that is becoming a global health problem with a growing prevalence. According to reports from the World Health Organization (WHO), the number of people with diabetes worldwide has soared in recent decades, and is expected to continue to rise in the foreseeable future. The disease not only impacts the quality of life of afflicted individuals, but also poses a huge economic burden to the global health system. With serious complications such as heart disease, kidney failure, and visual impairment, diabetes is one of the leading causes of morbidity and mortality in many countries.

Early identification of individuals at high risk of developing diabetes is a critical step in the prevention and management of this disease. Accurate predictions can enable early intervention,

lifestyle changes, and better medical management, all of which contribute to reducing the prevalence and impact of diabetes. In this context, the development of effective and reliable predictive models is crucial.

Technological advances in the field of artificial intelligence (AI), particularly Neural Network (NN) techniques, have opened up new opportunities in medical data analysis and predictive model development. Neural networks, with their ability to capture and model complex non-linear relationships in data, have shown promising results in various medical applications, including disease prediction. This technique enables large-scale data processing and analysis with high precision, which is particularly beneficial in addressing the challenges associated with diabetes prediction.

Python, as the dominant programming language in the field of data science and machine learning, offers a rich ecosystem with various libraries and tools that support the development of Neural Network models. Libraries such as TensorFlow, Keras, and PyTorch provide powerful and flexible frameworks for building, training, and optimizing NN models. In addition, Python has a large and active user community, which provides abundant support and resources

for developers and researchers.

This study aims to develop a predictive model for diabetes using a Neural Network implemented with Python. The data used in this study includes comprehensive clinical information, including risk factors such as age, gender, body mass index (BMI), blood pressure, blood glucose levels, and family history of diabetes. The model development stages include data pre-processing to ensure data quality and consistency, selection of relevant features to optimize model performance, model training using the NN algorithm, and model performance evaluation using appropriate metrics such as accuracy, sensitivity, specificity, and Area Under the Curve (AUC).

This technology-based approach is expected to not only improve the efficiency and accuracy of diabetes prediction, but also inspire further research using other AI techniques in medical applications. In the future, with bigger data and more sophisticated algorithms, the predictive models developed can continue to be refined and adapted to meet the increasingly complex and dynamic needs of health management.

2. METHODOLOGY

This research aims to develop a predictive model for diabetes using Neural Network (NN) with parameter optimization through genetic algorithm. The optimized parameters include the number of units in the hidden layer (one hidden layer) and the learning rate (α). This method consists of several main stages: Exploratory Data Analysis (EDA), pre- processing, Neural Network training stage, and neural network model evaluation. The following are the detailed steps of the method used:

2.1 Dataset

The research data used to calcify the likelihood of diabetes was obtained from the keagle website. The data used in this study consists of 768 rows of data and contains 8 relevant data attributes. The data consists of several possible factors such as Pregnancy, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age, with 1 attribute as the class, namely the Outcome attribute or diabetes outcome.

2.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is the initial step in data analysis that aims to understand characteristics, discover patterns, identify anomalies, and test assumptions with the help of statistics and visualization tools. EDA plays an important role in the data science process as it helps in gaining deep insights into the dataset before proceeding to the modeling and training stages.

2.3 Preprocesing data

The data will be checked first, starting from checking whether there is missing data and also whether there is duplicated data. The data will also be normalized using a standard scaler

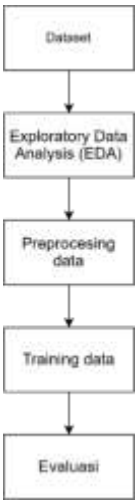
2.4 Training data

The training process of a classification model uses the available data to build a model that can perform classification. During the training process, the model will learn to find patterns or relationships between attributes in the data and the corresponding classes. After analyzing the training data, the model will find patterns that can be used to make predictions on data that has never been seen before.

2.5 Evaluation

Visualization of the model that has been created in the form of a neural network, which provides a visual description of the structure and decisions taken by the model. By viewing the model in the form of a neural network, we can easily understand the logic flow and decisions taken by the model when performing classification.

Figure 1. Research flow



3. THEORETICAL FOUNDATION

3.1 Python

Python is a programming language that is widely used in the field of data mining because of the many libraries that support data mining processing. Python has libraries such as Scikit-learn, pytorch that offer algorithms and functions to build neural network models and implement neural network algorithms based on existing data.

3.2 Pytorch

PyTorch is a popular open-source machine learning framework for deep learning model development. Developed by Facebook's AI

Research lab (FAIR), PyTorch provides great flexibility and ease of use in building, training, and deploying neural network models. With autograd support, GPU acceleration, and a large community, PyTorch remains the top choice for researchers and practitioners in building innovative AI solutions.

3.3 Neural Network

Neural Network (NN) is an artificial intelligence technique that mimics the way the human brain works to process information. Neural networks consist of layers of interconnected neurons, where each neuron sends signals to other neurons through connections that have certain weights. The neural network training process involves two main stages, namely feedforward and backpropagation.

3.4 Feedforward

Feedforward is the stage where input data is processed through the network to produce the output. The steps in the feedforward process are as follows :

a) Each input x_1 to x_i is multiplied by the weight of each connection and added with the neuron bias. Each neuron in the hidden layer receives signals from all neurons in the input layer. The total input value to neuron *j* in the hidden layer is calculated as:

$$z_j^{(1)} = \sum_{i=1}^{n} w_{ij}^{(1)} x_i + b_j^{(1)}$$

b) Output of neuron j in the hidden layer after ReLU activation function:

$$a_j^{(1)} = ReLU(z_j^{(1)}) = max(0, z_j^{(1)})$$

c) The output of the hidden layer becomes the input to the output layer.Total input values to neurons *k* at the output layer is calculated as:

$$z_k^{(2)} = \sum_{j=1}^{n} w_{jk}^{(1)} x_i + b_k^{(1)}$$

d) Output of the neuron k at the output layer after sigmoid activation function:

$$y_k = \sigma(z_k^{(2)}) = \frac{1}{1 + e^{-z_k^{(2)}}}$$

3.5 Backpropagation

Backpropagation is the training stage of a neural network to minimize the output error by updating the network weights. The steps in the backpropagation process are as follows:

a) Error is calculated as the difference between the generated output yk and target value tk.

$$\delta_k = y_k - t_k$$

b) The error in the hidden layer is calculated by transferring the error from the output layer

$$\delta_j^{(1)} = a_k^{(1)}(1 - a_j^{(1)}) \sum_k \delta_k w_{jk}^{(2)}$$

Since we are using the ReLU activation function in the hidden layer, the derivative is:

$$\alpha_j^{(1)} = \begin{cases} \sum_k \delta_k w_{jk}^{(2)}, if\ z_j^{(1)} > 0 \\ 0, \quad if\ z_j^{(1)} \leq 0 \end{cases}$$

c) Updating the weights and biases is done using the error gradient of the weights and biases.

$$\Delta w_{jk}^{(2)} = \alpha \delta_k z_j^{(1)}$$

$$b_k^{(2)} = b_k^{(2)} - \alpha \delta_k$$

$$w_{ij}^{(1)} = w_{ij}^{(1)} - \alpha \delta_j x_i$$

$$b_j^{(1)} = b_j^{(1)} - \alpha \delta_j$$

4. DISCUSSION

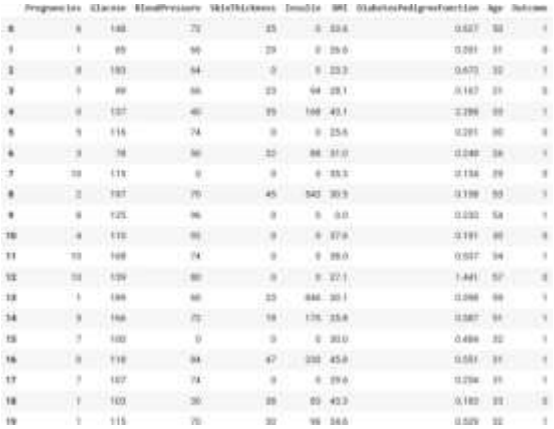
4.1 Dataset

The dataset in this study is taken from an online source, Kaggle, which is licensed CC0: Public Domain. The dataset consists of several attributes, including Pregnancy, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age.

To analyze and build the decision tree model, Python programming language and Google Colab development environment were used. Python is a programming language that is often used in data analysis and machine learning model building. Google Colab is a development environment that allows users to run Python code online and share projects easily.

Using the retrieved dataset and the Python development environment with Google Colab, this research aims to build a model. This model will be used to analyze and predict based on the attributes in the dataset.

Figure 2. Dataset



4.2 Exploratory Data Analysis (EDA)

In the Exploratory Data Analysis (EDA) process, the first step is to check the number of unique values in each feature. The check results show the following:

Pregnancies: 17 unique values

Glucose: 136 unique values

BloodPressure: 47 unique values

SkinThickness: 51 unique values Insulin: 186 unique values

BMI: 248 unique values

DiabetesPedigreeFunction: 517 unique values

Age: 52 unique values

Outcome: 2 unique values

To get an easier look at the data using histogram, pie chart, boxplot, and scatter plot visualizations.

Figure 3.1. Distribusi of Pregnancies

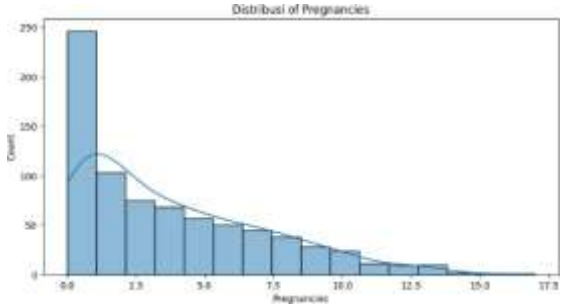


Figure 3.2. Distribusi of Glucose

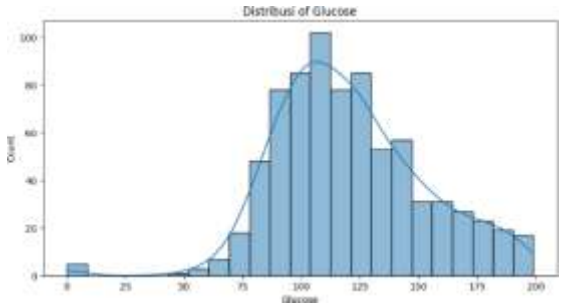


Figure 3.3. Distribusi of BloodPressure

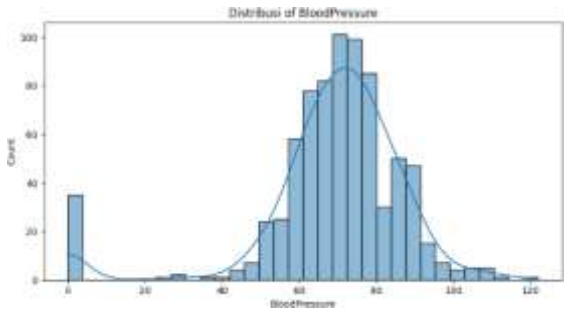


Figure 3.4. Distribusi of SkinThickness

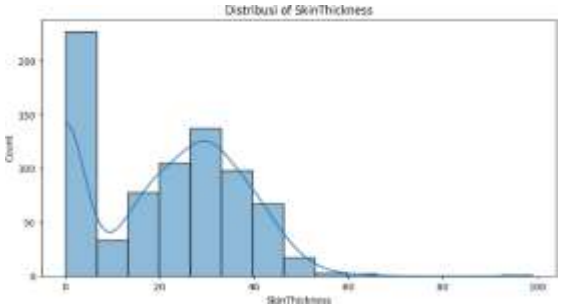


Figure 3.5. Distribusi of Insulin

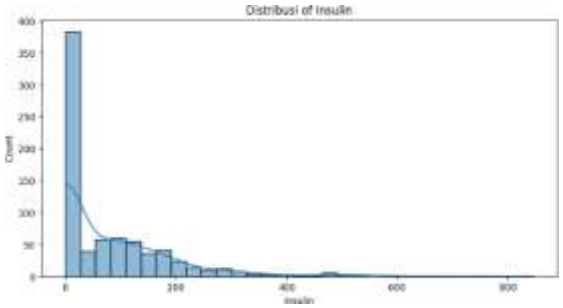


Figure 3.6. Distribusi of BMI

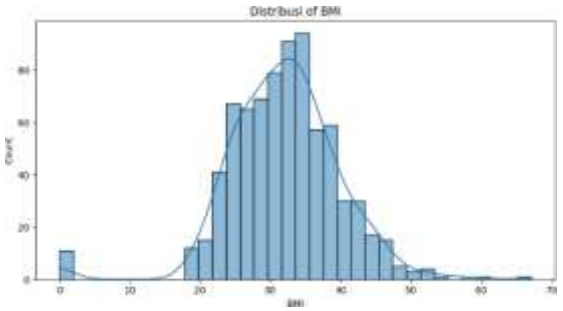


Figure 3.7. Distribusi of Diabetes Pedigree Function

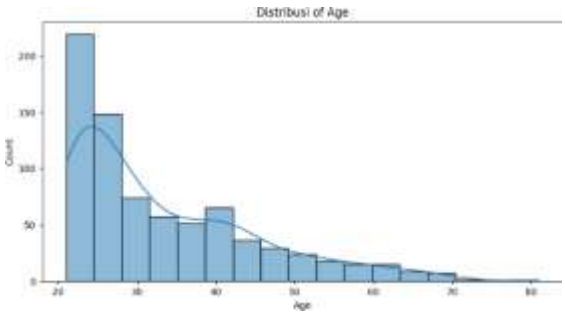
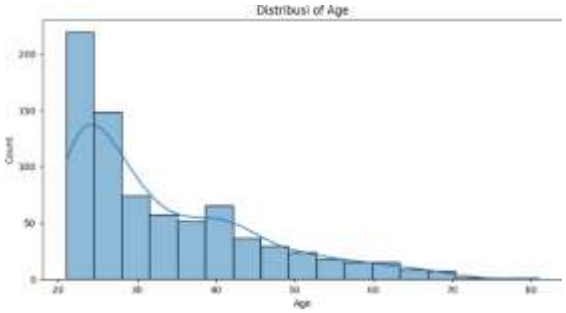
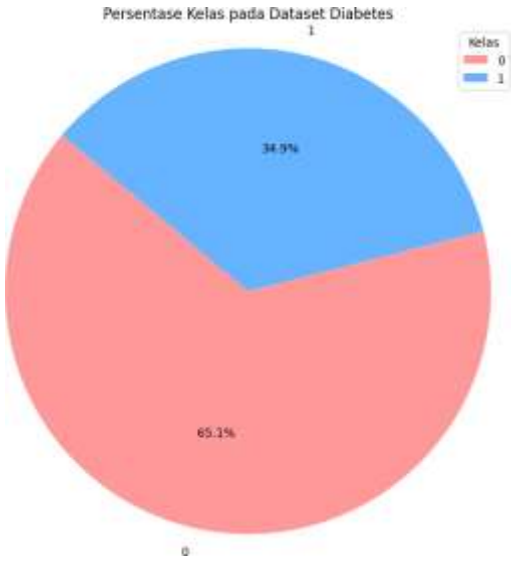


Figure 3.8. Distribusi of Age



In the distribution of the Pregnancies variable, the skewness was found to be positive with a value of 0.89991194, indicating a right-skewed distribution. Glucose distribution has a skewness close to normal distribution with a value of about 0.17341396. The BloodPressure, SkinThickness, and BMI distributions also showed near-normal distribution trends with skewness close to zero. However, in the BloodPressure distribution, the skewness was found to be negative with a value of around -1.84000523, indicating a left-skewed distribution. Whereas, in the Insulin distribution, the skewness is positive with a value of about 2.267, and in the Diabetes Pedigree Function distribution, the skewness is also positive with a value of about 1.91, indicating a right-skewed distribution. The Age distribution also has a positive skewness, but with a slightly lower value of about -1.127, indicating a right-skewed distribution.

Figure 4. Pie chart kelas diabetes



In the pie chart above, it is obtained that the target variable (diabetes) consists of two classes, namely class 0 (not affected by diabetes) of 65.1% and class 1 (affected by diabetes) of 34.9%. The use of boxplots allows for easier and clearer detection of outliers. These outliers can provide valuable insights into extreme variations in the data, which can affect interpretation and further analysis.

Figure 5.1. Boxplot of Pregnancies

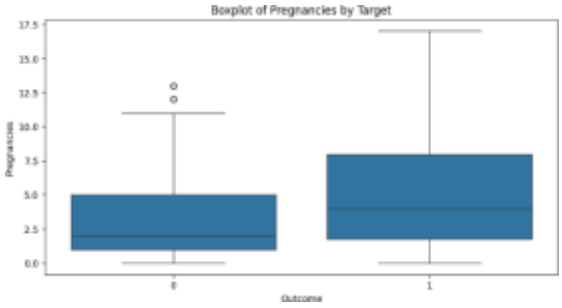


Figure 5.2. Boxplot of Glucose

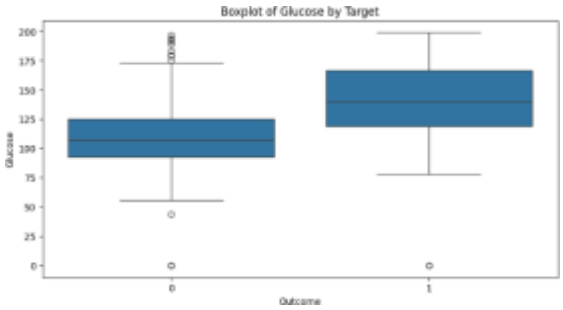


Figure 5.3. Boxplot of Blood Pressure

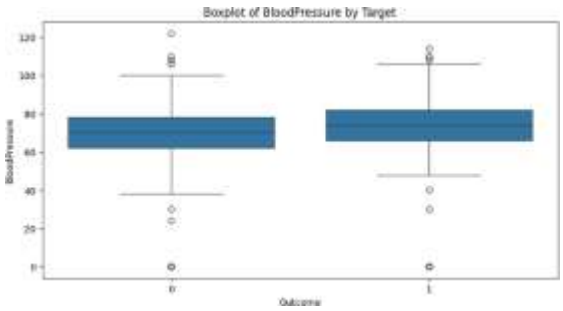


Figure 5.4. Boxplot of Skin Thickness

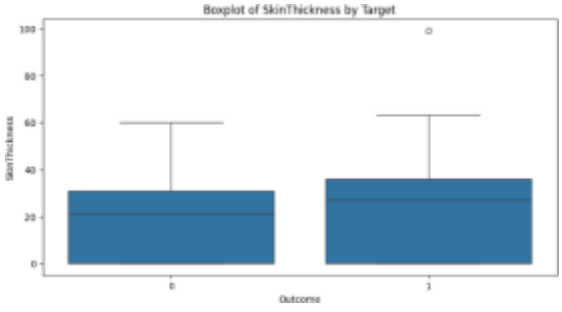


Figure 5.5. Boxplot of Insulin

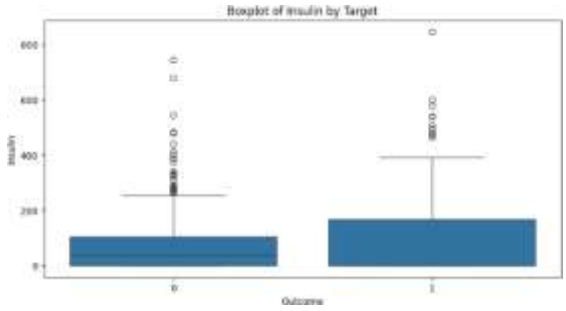


Figure 5.6. Boxplot of BMI

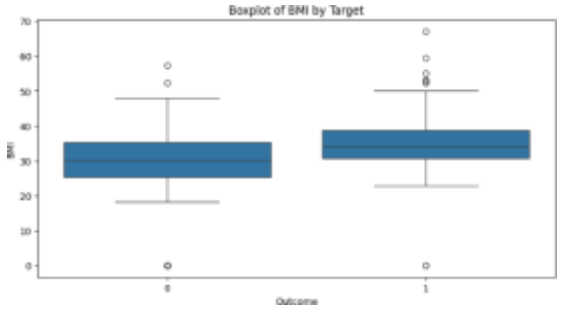


Figure 5.7. Boxplot of Diabetes Pedigree Function

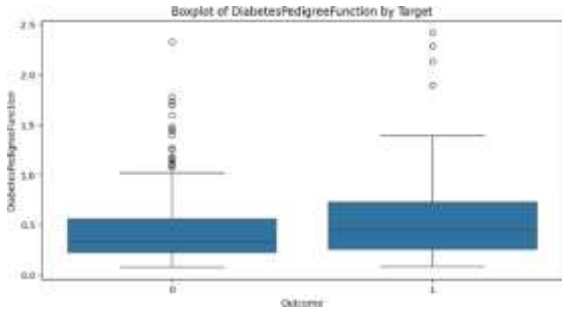
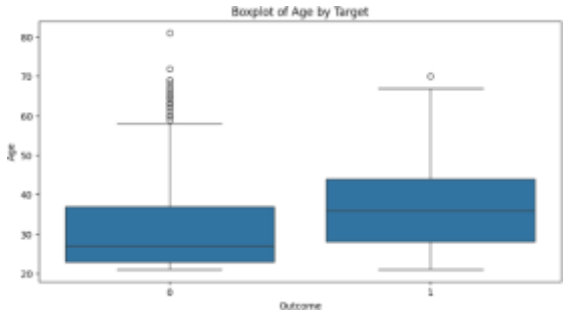
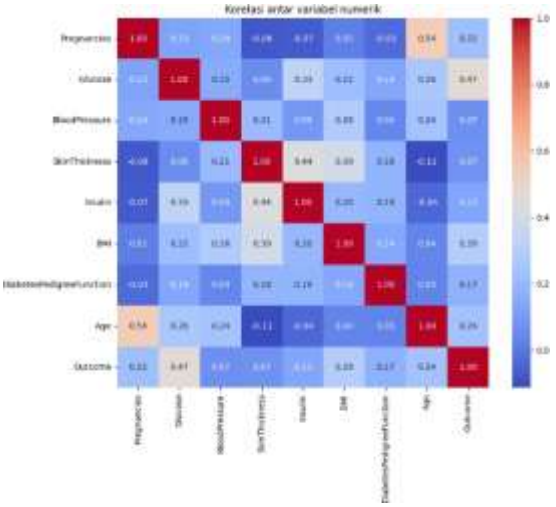


Figure 5.8. Boxplot of Age



Using heatmaps, it is easy to analyze the correlation patterns between various features. Heatmaps are an effective visual tool for understanding the relationships between variables in data. By analyzing the correlation between these features, we can identify relevant patterns and gain deeper insights into the data structure.

Figure 6. Heatmap



4.9 Data Preprocesing

Before training the model, the data is first cleaned through several important steps. The first step is the missing data check, where any entries that have missing values are identified and dealt with. The second step is the check and removal of duplicate data, which ensures that each entry in the dataset is unique and there are no unwanted repetitions.

Figure 7. Missing data

```
print(df.isnull().sum())
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Figure 8. Duplicate data

```
print("\nJumlah duplikat data:", df.duplicated().sum())
```

```
Jumlah duplikat data: 0
```

After the data has been cleaned, the next step is to separate the data into feature variables and target variables. All feature variables are then normalized using Standard Scaler. This normalization process aims to rescale the data so

that it has a distribution with a mean of zero and a standard deviation of one.

$$z_i = \frac{x_{i-\mu}}{\sigma}$$

The normalized data will be divided into two parts: training data and testing data using the train_test_split function, then X_train, X_test, y_train, y_test will be converted into tensor data.

Figure 9. Dataset splitting

```
X_train = torch.FloatTensor(X_train).to(device)
X_test = torch.FloatTensor(X_test).to(device)
y_train = torch.LongTensor(y_train).to(device)
y_test = torch.LongTensor(y_test).to(device)
```

4.4 Data Training

In the neural network training process, determining the network architecture is a crucial step. The network architecture used in this study consists of one input layer, two hidden layers, and one output layer. Each hidden layer uses a ReLU (Rectified Linear Unit) activation function, while the output layer uses a sigmoid activation function.

The use of ReLU activation function in the hidden layer aims to overcome the vanishing gradient problem and accelerate the convergence of the model. The sigmoid activation function in the output layer is used to convert the model output into probabilities, which is suitable for binary classification problems.

To prevent overfitting, a dropout regulation technique is applied. Dropout works by randomly removing neuron units within the network during each training iteration. This technique ensures that the network is not overly dependent on certain neurons, which can cause the model to overfit to the training data and lose generalization ability to new data.

Figure 10. Neural network architecture

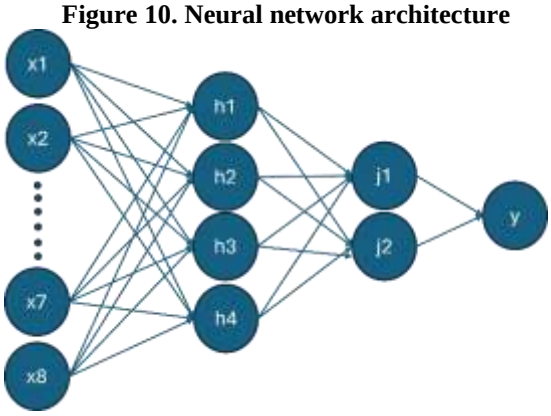


Figure 11. Neural network architecture (python)

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(8, 4)
        self.fc2 = nn.Linear(4, 2)
        self.fc3 = nn.Linear(2, 1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.relu(self.fc2(x))
        x = self.dropout(x)
        x = torch.sigmoid(self.fc3(x))
        return x
```

The loss used for binary classification uses the Binary Cross Entropy (BCE) loss function. BCE is an appropriate choice for binary classification problems as it calculates the loss between the predicted output value and the true value by considering the prediction probability.

$$L(y, y) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(y_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Where y is the true label and y is the predicted probability. In addition, to optimize the model training process, the Adam optimizer with weight decay is used. Adam's optimizer is a method that combines the advantages of AdaGrad and RMSProp optimization, so it can overcome the problem of sparse gradients and has adaptive capabilities for each parameter. The use of weight decay helps in additional regulation to prevent overfitting by adding a penalty to large weights in the loss function. The learning rate used is 0.01. This learning rate determines the size of the parameter update step in each training iteration.

Figure 12. Hyperparameter

```
model = NeuralNetwork().to(device)
criterion = nn.BCELoss()
optimizer = optim.AdamW(model.parameters(), lr=0.01)
```

The neural network model was trained with

a maximum of 10,000 epochs. To monitor the performance of the model during training, several evaluation variables are used, including: train_losses to record loss values on training data, test_losses to record loss values on testing data, and accuracies to record model accuracy on testing data. The variable best_val_loss is initialized with an infinite value to track the best loss value on the test data during the training process. In addition, the variable best_model_state is used to store the best state of the model obtained during training. The patience parameter is set to 5 to implement an early stopping mechanism, which will stop training if there is no improvement in the test loss within 5 consecutive epochs.

This approach ensures that model training can be stopped early if there is no significant improvement in performance, thus avoiding overfitting and efficient training time. The implementation of this evaluation variable aims to monitor and ensure that the resulting model has the best performance on the testdata.

Figure 13. Training data code

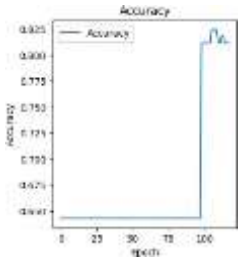
```
epochs = 10000
train_losses = []
test_losses = []
accuracies = []
best_val_loss = float('inf')
best_model_state = None
patience = 5
epochs_without_improvement = 0

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    y_pred = model(X_train).view(-1)
    y_train = y_train.float()
    loss = criterion(y_pred, y_train)
    loss.backward()
    optimizer.step()

    train_losses.append(loss.item())

    model.eval()
    with torch.no_grad():
        y_test_pred = model(X_test).view(-1)
        y_test =y_test.float()
        test_loss = criterion(y_test_pred, y_test)
        test_losses.append(test_loss.item())


        y_test_pred_binary = (y_test_pred > 0.5).float()
        accuracy = accuracy_score(y_test.cpu(), y_test_pred_binary.cpu())
        accuracies.append(accuracy)

    if epoch % 10 == 0:
        print(f'Epoch {epoch}/{epochs}, Train Loss: {loss.item():.4f}, Test

    if test_loss < best_val_loss:
        best_val_loss = test_loss
        best_model_state = model.state_dict()
        epochs_without_improvement = 0
    else:
        epochs_without_improvement += 1
        if epochs_without_improvement >= patience:
            print(f'Early stopping at epoch {epoch}')
            break
```

```
Epoch 0/10000, Train Loss: 0.6639, Test Loss: 0.6653, Accuracy: 0.6429
Epoch 10/10000, Train Loss: 0.6389, Test Loss: 0.6428, Accuracy: 0.6429
Epoch 20/10000, Train Loss: 0.6078, Test Loss: 0.6095, Accuracy: 0.6429
Epoch 30/10000, Train Loss: 0.5835, Test Loss: 0.5795, Accuracy: 0.6429
Epoch 40/10000, Train Loss: 0.5742, Test Loss: 0.5590, Accuracy: 0.6429
Epoch 50/10000, Train Loss: 0.5595, Test Loss: 0.5513, Accuracy: 0.6429
Epoch 60/10000, Train Loss: 0.5496, Test Loss: 0.5448, Accuracy: 0.6429
Epoch 70/10000, Train Loss: 0.5404, Test Loss: 0.5422, Accuracy: 0.6429
Epoch 80/10000, Train Loss: 0.5416, Test Loss: 0.5393, Accuracy: 0.6429
Epoch 90/10000, Train Loss: 0.5386, Test Loss: 0.5368, Accuracy: 0.6429
Epoch 100/10000, Train Loss: 0.5425, Test Loss: 0.5331, Accuracy: 0.8117
Epoch 110/10000, Train Loss: 0.5322, Test Loss: 0.5330, Accuracy: 0.8117
Early stopping at epoch 117
```

4.5 Evaluation

Based on the evaluation results, the neural network model performed quite well with an overall accuracy of 81.17%.

Figure 14. accuracy



The model has the same precision for both classes, which is 0.81. However, the recall for class 1 (0.62) is lower compared to class 0 (0.92), indicating that the model is more likely to correctly identify class 0 than class 1.

The F1-score for class 0 (0.86) is higher than that of class 1 (0.70), indicating that the balance between precision and recall is better for class 0. This may be due to the unbalanced distribution between the two classes, with more class 0 samples than class 1 (99 vs. 55).

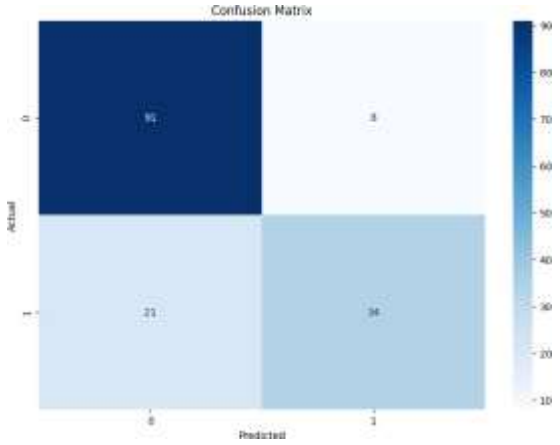
The macro average metric shows a precision of 0.81, recall of 0.77, and F1-score of 0.78. The weighted average metric gave a precision of 0.81, recall of 0.81, and F1-score of 0.80. This shows that the overall model performs well but needs to be improved in classifying the class 1 samples.

Figure 15. Classification Report

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.92      0.86        99
         1.0       0.81      0.62      0.70        55

    accuracy                           0.81       154
   macro avg       0.81      0.77      0.78       154
weighted avg       0.81      0.81      0.80       154

Accuracy: 0.8117
```

From this confusion matrix, it can be seen that the model correctly classified 91 class 0 samples (true negatives) and 34 class 1 samples correctly (true positives). However, the model also misclassified 8 class 0 samples as class 1 (false positives) and 21 class 1 samples as class 0 (false negatives).

Figure 16. Confusion Matrix

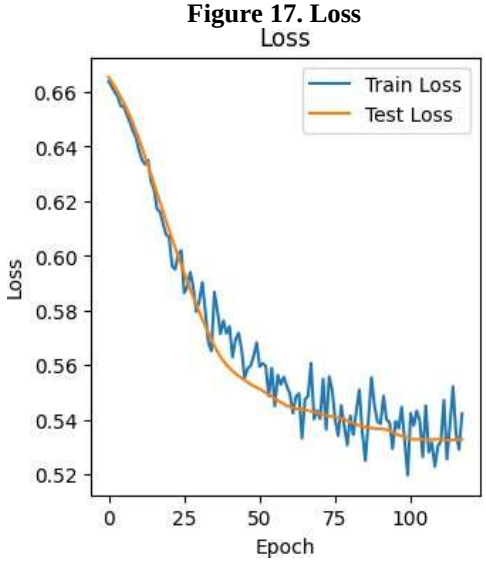


The loss on the test data is almost the same as the loss on the training data. This shows that the model has good generalization ability. In addition, the test loss was more stable and decreased significantly during the training process.

The stability and consistent decrease in the test loss indicate that the model does not experience overfitting, which means that the model is able to learn patterns from the training data without losing the ability to predict unseen data. In other words, the model's performance on training and testing data is balanced, so the model is expected to provide reliable predictions on new data.

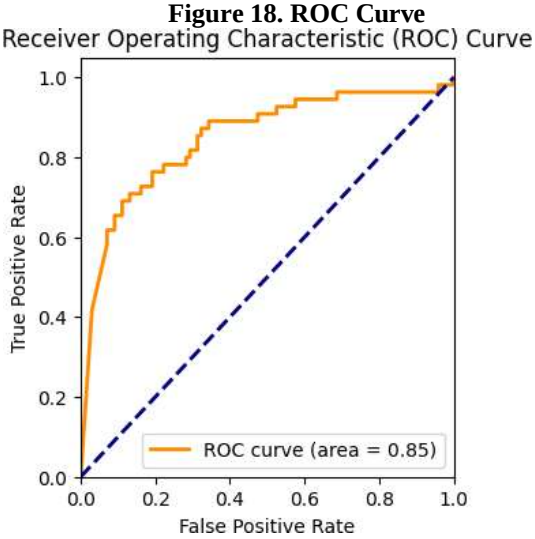
The good test loss reduction during training indicates that the model optimization is effective, and the model parameters are updated significantly, improving the model performance over time.

Figure 17. Loss



Based on the ROC curve analysis, the area under the curve (AUC) reached a value of 0.85. This indicates that the model has a good ability to distinguish between positive and negative classes. The AUC of 0.85 indicates that the model has a high degree of accuracy in predicting the correct class, where values close to 1 indicate perfect model performance and values of 0.5 indicate performance comparable to random guessing.

Figure 18. ROC Curve



5. CONCLUSION

Through the experiments and analysis conducted, it can be observed that the neural network model has managed to learn the data structure well, resulting in an impressive accuracy rate of 81.17%. The success of the model is not only limited to its ability to recognize the target classes, but also in distinguishing between negative and positive classes with a high degree of confidence, which is reflected in the ROC curve value of 0.85. These results show that the model has a good ability to map complex patterns in the data and capture important information relevant to diabetes classification.

The importance of applying several regularization techniques, such as dropout and L2 regularization, has been proven to prevent overfitting of the model. Thus, the model can be relied upon to provide consistent predictions and good generalization on data that has never been seen before. This confirms that the right approach in designing the neural network architecture, including the selection of the number of layers, the number of neurons, and the appropriate activation function, has a significant impact in improving the performance of the model.

Accurate prediction of diabetes risk can have a huge impact in health management and clinical decision-making. By using neural network technology, we can optimize the utilization of available medical data to provide valuable information for medical practitioners and patients.

REFERENCE

[1] Adam Mizza Zamani, Bilqis Amaliah, & Abdul Munif. (2012). Implementasi Algoritma Genetika pada Struktur Backpropagation Neural Network untuk Klasifikasi Kanker Payudara. Jurnal Teknik ITS, 1, 2301-9271.

[2] Sopiatul Ulum, Rizal Fahmi Alifa, Putri Rizkika, & Chaerur Rozikin. (2024). Perbandingan Performa Algoritma KNN dan SVM dalam Klasifikasi Kelayakan Air Minum. Generation Journal, 7(2), 141.

[3] Ardea Bagas Wibisono & Achmad Fahrurozi. (2024). Perbandingan Algoritma Klasifikasi dalam Pengklasifikasian Data Penyakit Jantung Koroner. Fakultas Teknologi Industri Universitas Gunadarma, Jl. Margonda Raya No. 100, Depok 16424, Jawa Barat.

[4] Mursyid Ardiansyah, Andi Sunyoto, & Emha Taufiq Luthfi. (2021). Analisis Perbandingan Akurasi Algoritma Naïve Bayes dan C4.5 untuk Klasifikasi Diabetes. Edumatic: Jurnal Pendidikan Informatika, 5(2), 147-156.

[5] Hana, F. M. (2024). "Klasifikasi Penderita Penyakit Diabetes Menggunakan Algoritma Decision Tree C4.5." Jurnal Kesehatan, 32, 123-135.

[6] Madaerdo, L., & Santoso, D. B. (2022). Perbandingan Algoritma KNN, Decision Tree, dan Random Forest pada Data Imbalanced Class untuk Klasifikasi Promosi Karyawan. (Tugas Akhir, Universitas Singaperbangsa Karawang).

[7] Hidayah, R. S. (2024). Klasifikasi Decision Tree Untuk Menentukan Kemungkinan Penyakit Stroke Dengan Bahasa Pemrograman Python. Jurnal Teknik Informatika Universitas Mercu Buana, 1(1), 1-10.

[8] Wahyuni, E. D., Arifiyanti, A. A., & Kustyani, M. (2019). Exploratory Data Analysis dalam Konteks Klasifikasi Data Mining. Dalam Prosiding Nasional Rekayasa Teknologi Industri dan Informasi XIV (pp. 263-269). ISSN: 1907-5995.

[9] Prajarini, D. (2016). Perbandingan Algoritma Klasifikasi Data Mining Untuk Prediksi Penyakit Kulit. INFORMAL, 1(3), 137. ISSN: 2503-250X.

[10] Azhar, Y., Firdausy, A. K., & Amelia, P. J. (2022). Perbandingan Algoritma Klasifikasi Data Mining Untuk Prediksi Penyakit Stroke. SINTECH JOURNAL, 5(2). ISSN: 2598-7305, E-ISSN: 2598-9642.

[11] Darsyah, M. Y. (2014). Pengunaan Stem and Leaf dan Boxplot untuk Analisis Data. JKPM, 1(1), 55. ISSN: 2339-2444.

[12] Sartika, D., & Sensuse, D. I. (2017). Perbandingan Algoritma Klasifikasi Naive Bayes, Nearest Neighbour, dan Decision Tree pada Studi Kasus Pengambilan Keputusan Pemilihan Pola Pakaian. Jatisi, 1(2), 151.

[13] Nasution, M. K., Saedudin, R. R., & Widartha, V. P. (2021). Perbandingan Akurasi Algoritma Naïve Bayes dan Algoritma XGBOOST pada Klasifikasi Penyakit Diabetes. e-Proceeding of Engineering, 8(5), 9765. ISSN: 2355-9365.

[14] Sinaga, S. H., Duha, A. A. M., & Banjarnahor, J. (Tahun). Analisis prediksi deteksi stroke dengan pendekatan EDA dan perbandingan algoritma machine learning. Jurnal Ilmiah Betrik, 1(2), 151-xxx. ISSN: 2339-1871.

[15] Putri, A. W. (2021). Implementasi artificial neural network (ANN) backpropagation untuk klasifikasi jenis penyakit pada daun tanaman tomat. MATHunesa, 9(2). e-ISSN: 2716-501X, p-ISSN: 2301-9115.

[16] Hadistio, R. R., Mawengkang, H., & Zarlis, M. (2022). Perbandingan Algoritma Stokastik Gradient Descent dan Naïve Bayes Pada Klasifikasi Retinopati Diabetik. Jurnal Media Informatika Budidarma, 6(1), 271-277.

[17] Sihombing, P. R., Suryadiningrat, Deden A. S., & Yuda, Y. P. A. C. (2022). Identifikasi Data Outlier (Pencilan) dan Kenormalan Data Pada Data Univarlat serta Alternatif Penyelesaiannya. Jurnal Ekonomi dan Statistik Indonesia, 2(3), 307-316.