Machine Learning CS462

Heart attack analysis and prediction using
Artificial Neural Network

May 21, 2024

Team Members: Islam Abd-Elhady Hassanein Mohamed
Enas Ragab Abdel-Latif Mohamed
Mariam Tarek Saad Mohamed

Contents

Chapter 1

Introduction/Executive Summary

1.1 Project Overview

The project addresses the critical global health issue of heart attacks by leveraging data analysis and machine learning techniques. Heart attacks are not only life-threatening but also impose significant healthcare burdens worldwide. The analysis focuses on identifying key factors linked to heart attacks and assessing their impact.

1.1.1 Dataset Features Description

- **Age**: The patient's age. (Continuous)
- **Sex**: The patient's gender (0 for female, 1 for male). (Categorical)
- **Chest Pain Type (cp)**: (Categorical)
  - **Value 0**: Typical Angina
  - **Value 1**: Atypical Angina
  - **Value 2**: Non-Anginal Pain
  - **Value 3**: Asymptomatic

- **Resting Blood Pressure (trtbps)**: (Continuous)
- **Serum Cholesterol Levels (chol)**: (Continuous)
- **Fasting Blood Sugar (fbs)**: (Categorical)
  - **Value 0**: ≤ 120 mg/dL
  - **Value 1**: > 120 mg/dL
- **Resting ECG Results (restecg)**: (Categorical)
  - **Value 0**: Normal
  - **Value 1**: ST-T Wave Abnormality
  - **Value 2**: Probable or Definite Left Ventricular Hypertrophy
- **Maximum Heart Rate During Exercise (thalachh)**: (Continuous)
- **Exercise-Induced Angina (exng)**: (Categorical)
  - **Value 0**: No
  - **Value 1**: Yes
- **ST-Segment Depression (oldpeak)**: (Continuous)
- **Slope of ST Segment (slp)**: (Categorical)
  - **Value 0**: Downsloping
  - **Value 1**: Flat
  - **Value 2**: Upsloping diagnosis
- **Number of Major Vessels Colored by Fluoroscopy (caa)**: (Categorical)
- **Thalassemia Type (thall)**: (Categorical)

  - **Value 0**: None (Normal)
  - **Value 1**: Fixed Defect
  - **Value 2**: Reversible Defect
  - **Value 3**: Thalassemia

- **Risk of Heart Attack (output)**: (Categorical)
  - **Value 0**: No
  - **Value 1**: Yes

1.2 Objective

The primary goal is to develop a predictive model using machine learning that can accurately determine the likelihood of an individual experiencing a heart attack. Such a model would be instrumental in enabling at-risk individuals to take preventative measures, potentially reducing the incidence and severity of heart attacks.

1.3 Problem Solvers and Algorithms

The analysis employs various data science methodologies, including data preprocessing, exploratory data analysis (EDA), and machine learning. The EDA provides insights into the data through univariate, bivariate, and multivariate analysis, revealing patterns and relationships that are crucial for the subsequent modeling phase.

1.4 Selected Algorithm

The chosen algorithm for building the predictive model is Artificial Neural Networks (ANN). ANNs are well-suited for this type of classification task because of their ability to model complex nonlinear relationships and interactions between variables. This makes them particularly effective in handling the multifaceted nature of medical data related to heart health.

This summary encapsulates the essence of the project, highlighting the integration of advanced analytical techniques to tackle a significant health issue. By using ANN, the project aims to create a robust model that aids in early detection and prevention strategies for heart attacks.

Chapter 2

Methodology

2.1 Description of Artificial Neural Network Algorithms

Artificial Neural Networks (ANNs) are a foundational element of modern machine learning, inspired by the biological neural networks that constitute animal brains. They are particularly well-suited to tasks that involve recognizing patterns or making predictions based on complex, non-linear inputs.

2.1.1 Basic Concept

An ANN consists of nodes (neurons) and edges (synapses) that connect these nodes. Each node in one layer is connected to nodes in the next layer through edges that carry weights. The neurons process inputs and produce outputs based on an activation function.

2.1.2 Architecture

1. **Input Layer**: This layer consists of neurons that receive the input features. Each neuron corresponds to one feature in the input data.

2. **Hidden Layers**: These layers perform most of the computational work through their neurons. Each neuron in a hidden layer transforms values from the previous layer with a weighted sum followed by a non-linear activation function. The number and size of hidden layers and neurons can vary and significantly influence the network's capability.

3. **Output Layer**: The final layer that outputs the prediction of the network. For classification tasks, this layer typically has as many neurons as there are classes; for regression tasks, it usually contains a single neuron.

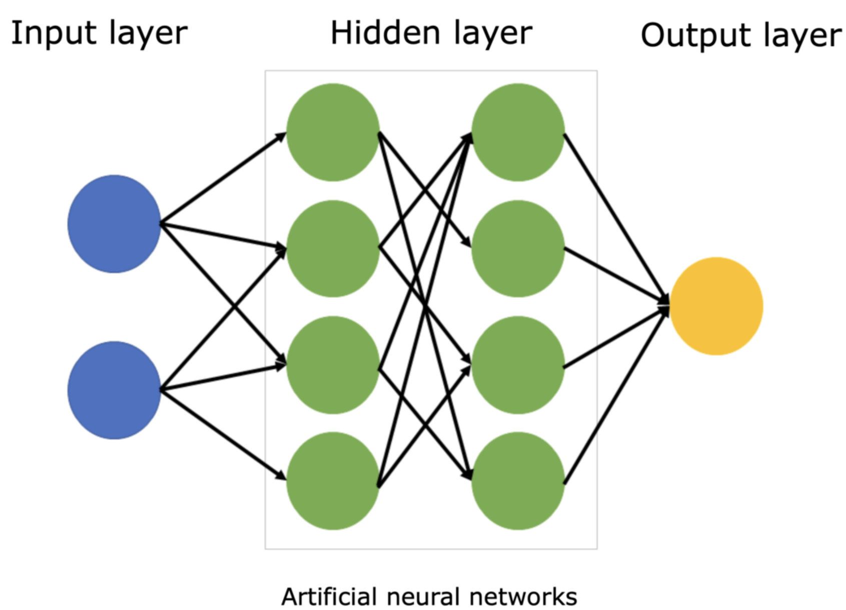


Figure 2.1: Artificial Neural Network Architecture

2.1.3 Algorithms

Feedforward

The process of feeding input through the layers to the output is known as feedforward. Each neuron receives an input, applies a weight to it, adds a bias, and uses an activation function to produce an output. The formula for a neuron's output is:

$$y = f\left(\sum(w_i \cdot x_i) + b\right)$$

where x_i are the inputs, w_i are the weights, b is the bias, and f is the activation function (e.g., sigmoid, ReLU).

Backpropagation

Backpropagation is the fundamental algorithm for training ANNs, consisting of a forward pass and a backward pass. In the forward pass, input data is passed through the network to generate output. During the backward pass, the network's output is compared to the desired output, and the error is calculated. This error is then propagated back through the network, layer by layer, updating the weights to minimize the error. The most common method for weight update is stochastic gradient descent (SGD), although other optimizers like Adam and RMSprop are often used for better performance.

Training Process

1. **Initialization**: Assign random weights and biases to all neurons.
2. **Forward Pass**: Compute the output of each neuron from the input layer to the output layer.

3. **Loss Calculation**: Calculate the error at the output.

4. **Backward Pass**: Use backpropagation to compute the gradient of the error with respect to each weight and bias.

5. **Weight Update**: Adjust the weights and biases based on the gradients to minimize the loss.

2.1.4 Activation Functions

Activation functions introduce non-linear properties to the network which allows the model to learn more complex patterns. Common activation functions include:

- **ReLU (Rectified Linear Unit)**: Provides a very efficient and simple non-linearity, allowing models to converge quickly and maintain effective performance.

- **Sigmoid**: Maps the input values to a range between 0 and 1, useful for binary classification.

- **Tanh**: Similar to sigmoid but maps values between -1 and 1.

- **Softmax**: Used in the output layer of multi-class classification problems. It turns logits (raw prediction values) into probabilities by taking the exponential of each output and then normalizing these values by dividing by the sum of all exponentials.

2.1.5 Applications

ANNs have broad applications across many fields, including speech recognition, image recognition, medical diagnosis, and more, due to their ability to perform well with large and complex datasets.

2.1.6 Limitations

Despite their flexibility and power, ANNs can be prone to overfitting, especially with small datasets. They also require significant computational resources, particularly for large networks and datasets.

2.2 Main Features of Artificial Neural Networks

Artificial Neural Networks (ANNs) are versatile and powerful computational models that have become fundamental to various applications in machine learning and artificial intelligence. Here are the main features that distinguish ANNs:

1. **Layered Structure**
    - **Input Layer:** Receives raw input data.
    - **Hidden Layers:** Perform computations and feature transformations. The number of hidden layers and the number of neurons in each layer can vary, defining the complexity and depth of the network.
    - **Output Layer:** Produces the final output of the network, which could be a class label in classification tasks or a continuous value in regression.
2. **Connection Weights**
    - Each connection between two neurons has an associated weight that adjusts as the network learns.

The weight adjusts during the training phase to minimize the error in predictions.

3. **Activation Functions**
    - Neurons use non-linear activation functions to compute their output signals. Common functions include sigmoid, ReLU (Rectified Linear Unit), and softmax. These functions help the network learn complex patterns in the data.
4. **Learning Ability**
    - ANNs can learn and model non-linear and complex relationships, making them very effective for problems like image recognition, natural language processing, and many others. They adjust their model parameters (weights) based on the error of the output compared to the expected result.
5. **Backpropagation**
    - This is the key training algorithm for ANNs, where the network learns from the errors by propagating them back through the network layers. It adjusts the weights to minimize these errors using gradient descent or other optimization methods.
6. **Adaptability**
    - ANNs can adapt to new data without needing explicit reprogramming. This feature is particularly useful in dynamic environments where the characteristics of the input data can change over time.

7. **Fault Tolerance**
    - Due to their parallel architecture, neural networks are relatively fault tolerant. The degradation of performance tends to be gradual as some of the neurons or connections fail, rather than leading to complete failure.
8. **Generalization**
    - Neural networks have the ability to generalize from their training data and perform well on new, unseen data. This is crucial for practical applications where the exact inputs during training may not represent all variations during testing.
9. **Scalability and Integration**
    - ANNs are inherently scalable, capable of handling larger and more complex datasets by increasing the network size (more layers or more neurons). They are also easy to integrate with other data processing and analysis techniques.
10. **Parallel Processing**
    - Neural networks are particularly well-suited for parallel processing, enabling them to manage large volumes of data and complex models efficiently, especially when implemented on modern hardware like GPUs.

2.3 Artificial Neural Network flowchart and pseudocode

2.3.1 Flowchart

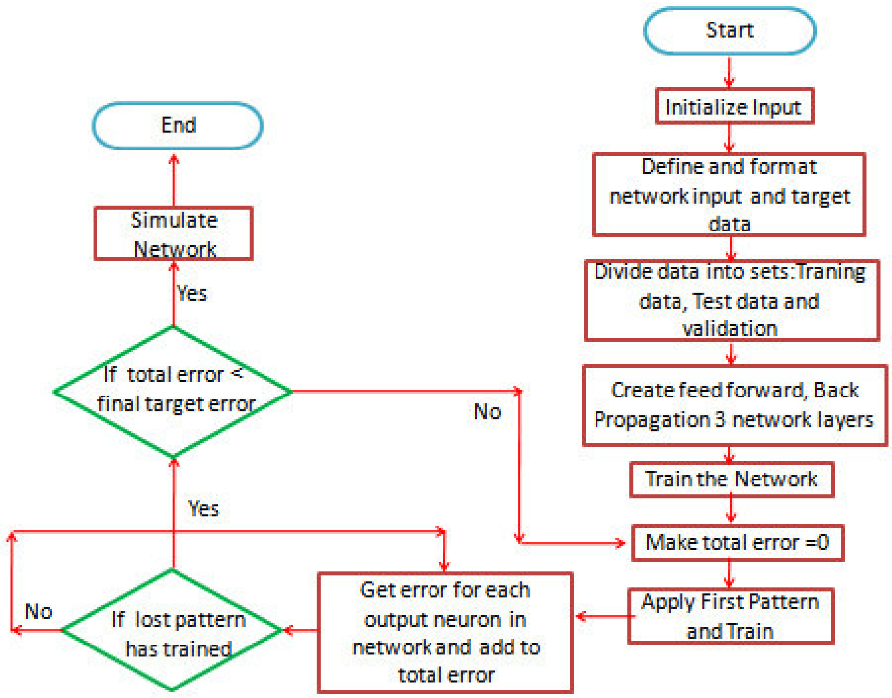


Figure 2.2: Flowchart for Artificial Neural Network (ANN)

The flowchart outlines the training process of an artificial neural network (ANN). The steps are as follows:

1. **Start:** The process begins with initializing the network.

2. **Initialize Input:**

    - *Define and format network input and target data:* Prepare the data that will be input into the network, along with the expected output data (target).

   - *Divide data into sets:* Split the prepared data into training, testing, and validation datasets.

3. **Create Feed Forward, Back Propagation 3 Network Layers:** Establish the architecture of the neural network, which includes setting up the layers and the method for back propagation.

4. **Train the Network:** Begin the training phase where the network learns from the training data by adjusting the weights of the network.

   - *Apply First Pattern and Train:* Start with the first pattern from the training set and perform training steps.
   - *Get Error for Each Output Neuron in Network and Add to Total Error:* After applying a pattern, calculate the error at each output neuron and aggregate it into a total error.

5. **Make Total Error = 0:** Reset the total error after processing all patterns, which usually signifies a new iteration over the training dataset.

6. **Simulate Network:** After training, simulate the network with new data to verify its performance.

7. **If Total Error < Final Target Error:** Check if the total error is below a predefined threshold (final target error), which determines if the training is sufficient.

- *Yes:* Proceed to end the training process.
- *No:* Continue the training by returning to apply the first pattern and retraining the network.

8. **If Lost Pattern has Trained:** This decision point likely checks whether all patterns have been successfully learned by the network.
    - *Yes:* End the training process if the network has learned all patterns to satisfaction.
    - *No:* Continue with training if some patterns are not learned well.

9. **End:** Conclude the training process once the network performs satisfactorily according to the defined criteria.

This flowchart is typical for a training cycle in machine learning, particularly in neural networks, focusing on iterative learning and error reduction to improve accuracy.

2.3.2 Pseudocode

Writing pseudocode for an Artificial Neural Network (ANN) involves detailing the steps for initializing the network, conducting forward propagation, backpropagation, and updating weights. Below is a simplified pseudocode that captures the general workflow of training an ANN for tasks like classification or regression:

Algorithm 1 Artificial Neural Network Training and Prediction

```
Initialize ANN:
Initialize weights and biases with small random values
Define number of layers and number of neurons per layer
Choose activation function for each layer

function ForwardPropagation(Input)
    for each layer in the network do
        Calculate net input: net_input = weights × inputs + biases
        Apply activation function to net_input to get layer_output
        Set inputs for next layer = layer_output
    end for
    return output of the final layer
end function

function CalculateLoss(PredictedOutput, TrueOutput)
    Use loss function (e.g., MSE for regression, cross-entropy for classification)
    Compute loss to measure discrepancy between PredictedOutput and TrueOutput
    return Loss
end function

function Backpropagation(Loss)
    Calculate gradients of Loss w.r.t. each weight and bias
    for each layer from last to first do
        Compute gradient of Loss w.r.t. weights and biases
        Update weights and biases: new_weight = old_weight − learning_rate × gradient
    end for
end function

function TrainNetwork(TrainingData, Epochs, LearningRate)
    for each epoch do
        for each example in TrainingData do
            Output ← ForwardPropagation(example.Input)
            Loss ← CalculateLoss(Output, example.TrueOutput)
            Backpropagation(Loss)
        end for
        Optionally evaluate performance on validation set
    end for
end function

function PredictNewData(Input)
    Output ← ForwardPropagation(Input)
    return Output
end function
```

2.4 The time complexity of Artificial Neural Network

2.4.1 Factors Affecting Time Complexity

Time complexity in ANNs refers to how the computation time increases based on factors such as the number of neurons, number of layers, and the size of the input data. Important factors include:

- **Number of Neurons and Layers:** More neurons and layers increase the number of computations.
- **Connectivity:** Dense connections increase operations, while sparse connections (like in convolutional neural networks) can reduce complexity.
- **Number of Training Examples:** Directly impacts the training time as each example requires computation.
- **Batch Size:** Influences the number of weight updates and calculations per epoch.
- **Number of Epochs:** More epochs result in more data passes, increasing total computation time.

2.4.2 Example and Analysis

Consider a simple fully connected feedforward neural network:

- **Input Layer:** 100 neurons
- **Hidden Layer:** 50 neurons
- **Output Layer:** 10 neurons

Forward Pass Complexity

The computation involves:

- **Input to Hidden:** $100 \times 50 = 5000$ operations.
- **Hidden to Output:** $50 \times 10 = 500$ operations.

Total operations for one forward pass: $5000 + 500 = 5500$ operations.

Backward Pass Complexity

Similar to the forward pass, the backward pass involves propagating errors back through the network, requiring approximately the same number of operations.

Total Complexity Per Epoch

If N is the number of training examples, the total operations per epoch would be $N \times 5500$.

2.4.3 Big O Notation

The time complexity T for training an ANN can be approximated by:

$$O(E \times N \times L \times s \times t)$$

where E is the number of epochs, N is the number of training examples, L is the number of layers, s is the average number of synapses per neuron, and t is the time complexity per synapse per example.

2.4.4 Conclusion

The time complexity of ANNs can be significant, especially for large, densely connected networks. Optimizations such as using GPUs and efficient network architectures are crucial for managing this complexity.

2.5 Description of the Artificial Neural Network Using K-Fold Cross-Validation used in this project

2.5.1 Model Description

Architecture

The neural network is structured as a sequential model comprising several layers:

1. **Input Layer:** A dense layer with 32 neurons, using the ReLU (Rectified Linear Unit) activation function. The input shape is determined by the number of features in the training data.
2. **Hidden Layer:** Another dense layer with 16 neurons, also using the ReLU activation function.
3. **Output Layer:** A dense layer with a single neuron, using the sigmoid activation function, suitable for binary classification tasks.

Compilation

The model is compiled with the following configurations:

- **Loss Function:** Binary cross-entropy, appropriate for binary classification problems.
- **Optimizer:** Adam optimizer, known for its efficiency in handling sparse gradients and adaptive learning rate capabilities.
- **Metrics:** Accuracy, to evaluate the model's performance during training and validation.

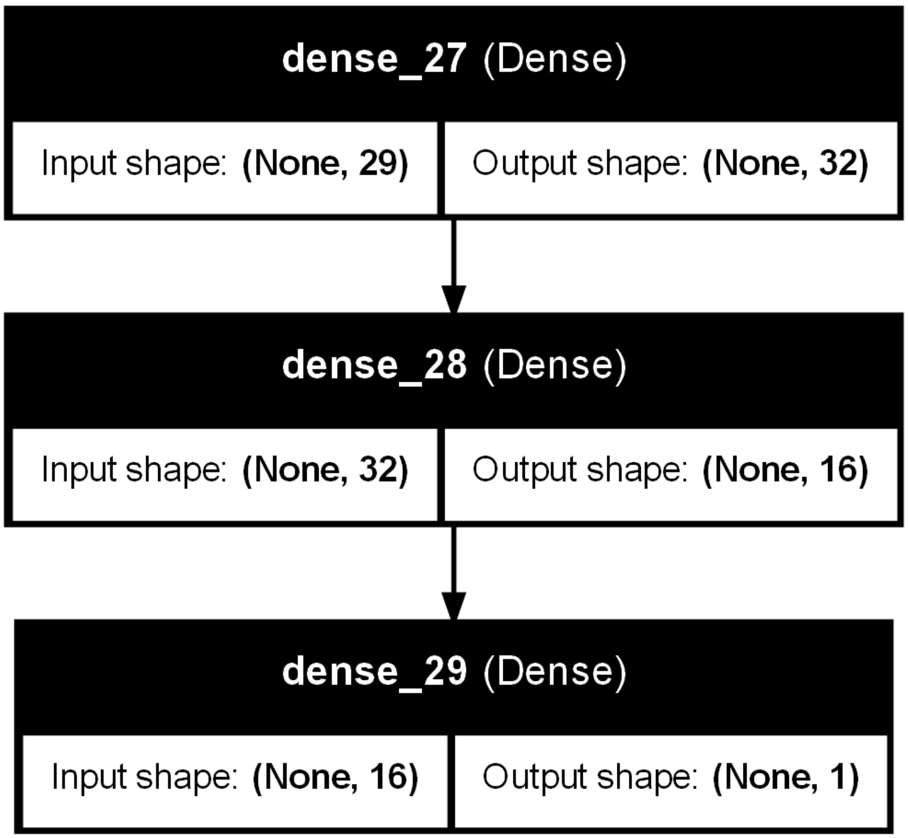


Figure 2.3: Artificial Neural Network Architecture used in this project

Training

The training process incorporates K-fold cross-validation with 5 splits, enhancing the model's robustness and generalizability:

- **Early Stopping:** A callback that stops training when the validation loss does not decrease for 10 consecutive epochs, preventing overfitting and ensuring generalization.
- **Epochs:** Up to 100 training epochs, though early stopping may halt training sooner.
- **Validation:** Each training epoch is validated on a respective validation fold to monitor performance on unseen data.
- **Verbosity:** Set to 0 to suppress output during the training process.

Evaluation

After training on each fold, the model is evaluated on the corresponding validation set:

- **Accuracy for each fold:** Displays how well the model performed on each validation fold.
- **Average Accuracy:** Calculated from the accuracies of all folds, providing an overall metric of model performance across the cross-validation process.

2.5.2 Model Code

Build ANN Model Training with Cross-Validation:

```python
# Set the random seed for reproducibility
tf.random.set_seed(42)
# Define the number of folds for KFold cross-validation
n_splits = 5
kfold = KFold(n_splits=n_splits, shuffle=True, random_state=42)
# Prepare to collect scores and histories
accuracies = []
all_histories = []

# KFold Cross Validation
for train_index, val_index in kfold.split(X_train):
    # Split data
    X_train_kfold, X_val_kfold = X_train[train_index], X_train[val_index]
    y_train_kfold, y_val_kfold = y_train[train_index], y_train[val_index]

    # Create a new instance of the model (to reinitialize weights)
    ANN_model = tf.keras.Sequential([
        tf.keras.layers.Dense(32, activation="relu", input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(16, activation="relu"),
        tf.keras.layers.Dense(1, activation="sigmoid")
    ])

    # Compile the model
    ANN_model.compile(loss="binary_crossentropy",
                      optimizer=tf.keras.optimizers.Adam(),
                      metrics=["accuracy"])

    # Early stopping callback
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    )

    # Fit the model
    history = ANN_model.fit(X_train_kfold, y_train_kfold,
                            epochs=100,
                            validation_data=(X_val_kfold, y_val_kfold),
                            callbacks=[early_stopping],
                            verbose=0)  # Set verbose to 0 to reduce output

    # Collect the history from each fold
    all_histories.append(history)

    # Evaluate the model on the validation set
    scores = ANN_model.evaluate(X_val_kfold, y_val_kfold, verbose=0)
    accuracies.append(scores[1])  # Assume that the accuracy is the second metric
# Print the accuracy for each fold
print("Accuracy for each fold:", accuracies)
# Print the average accuracy
print("Average accuracy:", np.mean(accuracies))
```

2.5.3 Summary

This setup ensures robust analysis and prediction on unseen data, optimizing the training process with mechanisms such as early stopping, making the model both efficient and effective.

Chapter 3

Experimental Simulation

3.1 Introduction

This chapter outlines the development and evaluation of an Artificial Neural Network (ANN) model aimed at predicting heart attack occurrences. It covers the programming languages, environments used, and detailed discussions on the model's primary functions, testing, and parameter settings.

3.2 Programming Languages and Environments

3.2.1 Tools and Libraries

Exploratory Data Analysis (EDA) Libraries

- **Pandas**: Essential for manipulating numerical tables and time series.
- **NumPy**: Supports large, multi-dimensional arrays and matrices, along with high-level mathematical functions.
- **Plotly**: Used for creating interactive plots.

- **Seaborn and Matplotlib**: For drawing attractive and informative statistical graphics.
- **Plotly Figure Factory and Subplots**: Combine multiple plots into a single figure.

Data Preprocessing Libraries

- **datasist**: Quick and easy functions for typical data analysis and feature engineering tasks.
- **Sklearn**: Includes tools for machine learning and statistical modeling.
- **Imbalanced-learn**: Deals with imbalanced data sets.
- **Category Encoders**: Encodes categorical variables into quantitative variables.
- **SimpleImputer**: Handles missing data by imputing missing values.

Machine Learning and Deep Learning Libraries

- **TensorFlow**: Platform for creating machine learning models, particularly deep learning models.
- **Sklearn Feature Selection**: Select features based on various criteria.
- **Imblearn Pipeline**: Automates machine learning workflows with integrated balancing techniques.
- **TensorFlow Keras Utilities**: Efficient tools for building and training neural network models.

Miscellaneous

- **warnings**: Used to suppress warnings to ensure cleaner output.

3.2.2 Development Environment

- **Interactive Development**: Utilizes Python within Jupyter Notebooks for interactive code execution.
- **Visualization Support**: Emphasis on visual data exploration integrated within Jupyter Notebooks.
- **Version Control and Sharing**: Notebooks can be version-controlled and shared via platforms like GitHub or in formats such as HTML or PDF.

3.3 Primary Function and Procedures

3.3.1 Data Preprocessing

Data is prepared using techniques such as scaling, encoding, and sampling to make it suitable for training the ANN model.

3.3.2 Model Building and Training

The ANN model is built and trained using TensorFlow's Keras API, involving:

- Defining the model architecture.
- Configuring the learning process with cross-validation.
- Evaluating model performance using accuracy, F1-score, and ROC curves.

3.4 Testing the Programmed Codes

3.4.1 Test Dataset

A subset of the data, reserved for testing, is used to assess the model's generalization capability.

3.4.2 Performance Metrics

The model's performance is evaluated using various metrics including accuracy, F1-score, and confusion matrices.

3.5 Setting Program Parameters and Constants

3.5.1 Model Parameters

Parameters such as the number of layers, neurons, activation functions, and optimizer settings are specified.

3.5.2 Training Parameters

Training settings including the number of epochs, batch size, and validation strategies are discussed.

3.6 Conclusion

This project illustrates the application of deep learning to predict health outcomes, emphasizing the importance of precise parameter tuning and robust model evaluation.

Chapter 4

Results and Technical Discussion

4.1 Report the main program results and outputs

When using the model with the Artificial Neural Network algorithm, the following results were produced:

The best accuracy rate: 88.5%

4.1.1 Run the ANN model

The model was run more than once and we obtained the following results:

```
# Print the accuracy for each fold
print("Accuracy for each fold:", accuracies)

# Print the average accuracy
print("Average accuracy:", np.mean(accuracies))
```

```
Accuracy for each fold: [0.8367347121238708, 0.8333333134651184, 0.7708333134651184, 0.9375, 0.7916666865348816]
Average accuracy: 0.8340136051177979
```

Figure 4.1: Results of training the model for the 1st time

```python
# Evaluate model on the test dataset
loss, accuracy = ANN_model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8386 - loss: 0.3715
Test accuracy: 0.8360655903816223
```

Figure 4.2: Test accuracy for 1st time

```python
# Print the accuracy for each fold
print("Accuracy for each fold:", accuracies)

# Print the average accuracy
print("Average accuracy:", np.mean(accuracies))
```

```
Accuracy for each fold: [0.8163265585899353, 0.7916666865348816, 0.8125, 0.9166666865348816, 0.75]
Average accuracy: 0.8174319863319397
```

Figure 4.3: Results of training the model for the 2nd time

```python
# Evaluate model on the test dataset
loss, accuracy = ANN_model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8818 - loss: 0.3599
Test accuracy: 0.8852459192276001
```

Figure 4.4: Test accuracy for 2nd time

```python
# Print the accuracy for each fold
print("Accuracy for each fold:", accuracies)

# Print the average accuracy
print("Average accuracy:", np.mean(accuracies))
```

```
Accuracy for each fold: [0.8367347121238708, 0.7708333134651184, 0.7916666865348816, 0.9583333134651184, 0.8125]
Average accuracy: 0.8340136051177979
```

Figure 4.5: Results of training the model for the 3rd time

```
# Evaluate model on the test dataset
loss, accuracy = ANN_model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8600 - loss: 0.3769
Test accuracy: 0.8524590134620667
```

Figure 4.6: Test accuracy for the 3rd time

```
# Print the accuracy for each fold
print("Accuracy for each fold:", accuracies)

# Print the average accuracy
print("Average accuracy:", np.mean(accuracies))
```

```
Accuracy for each fold: [0.8367347121238708, 0.875, 0.75, 0.8958333134651184, 0.75]
Average accuracy: 0.8215136051177978
```

Figure 4.7: Results of training the model for the 4th time

```
# Evaluate model on the test dataset
loss, accuracy = ANN_model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8069 - loss: 0.4132
Test accuracy: 0.8196721076965332
```

Figure 4.8: Test accuracy for the 4th time

```
# Print the accuracy for each fold
print("Accuracy for each fold:", accuracies)

# Print the average accuracy
print("Average accuracy:", np.mean(accuracies))
```

```
Accuracy for each fold: [0.8163265585899353, 0.8125, 0.75, 0.875, 0.75]
Average accuracy: 0.800765311717987
```

Figure 4.9: Results of training the model for the 5th time

```
# Evaluate model on the test dataset
loss, accuracy = ANN_model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

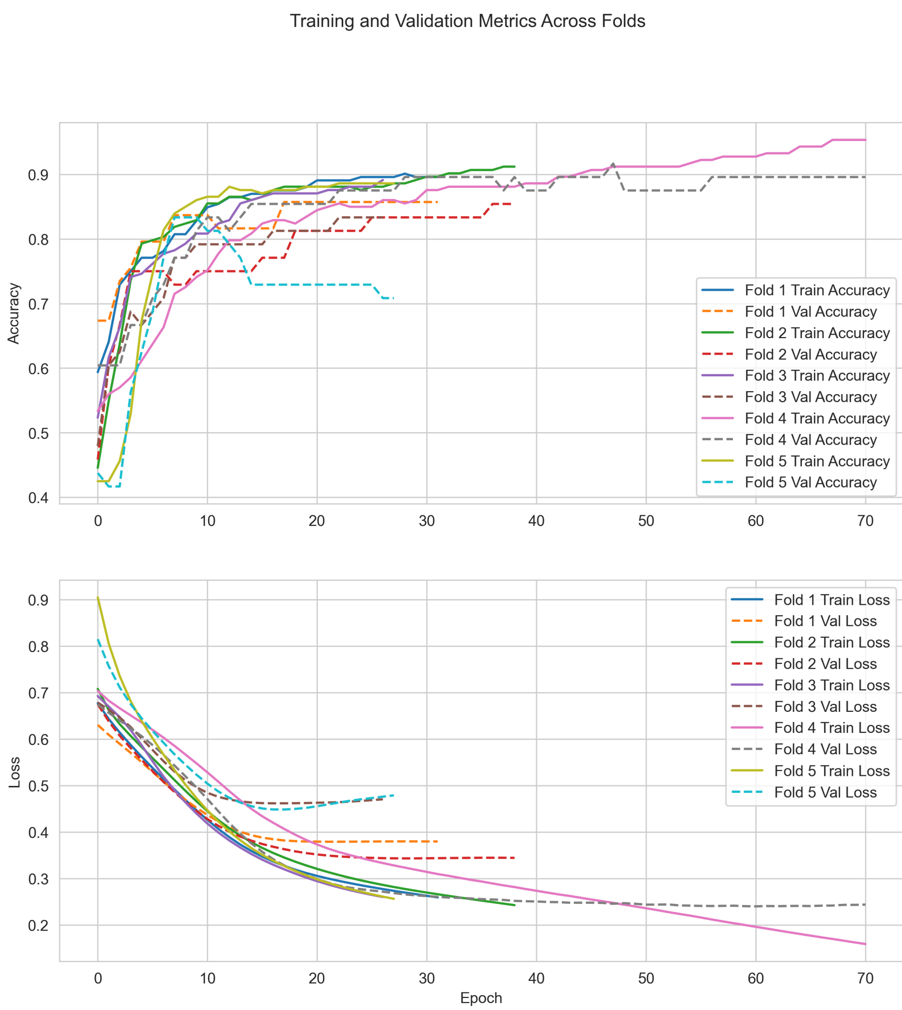
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8600 - loss: 0.3883
Test accuracy: 0.8524590134620667
```

Figure 4.10: Test accuracy for the 5th time

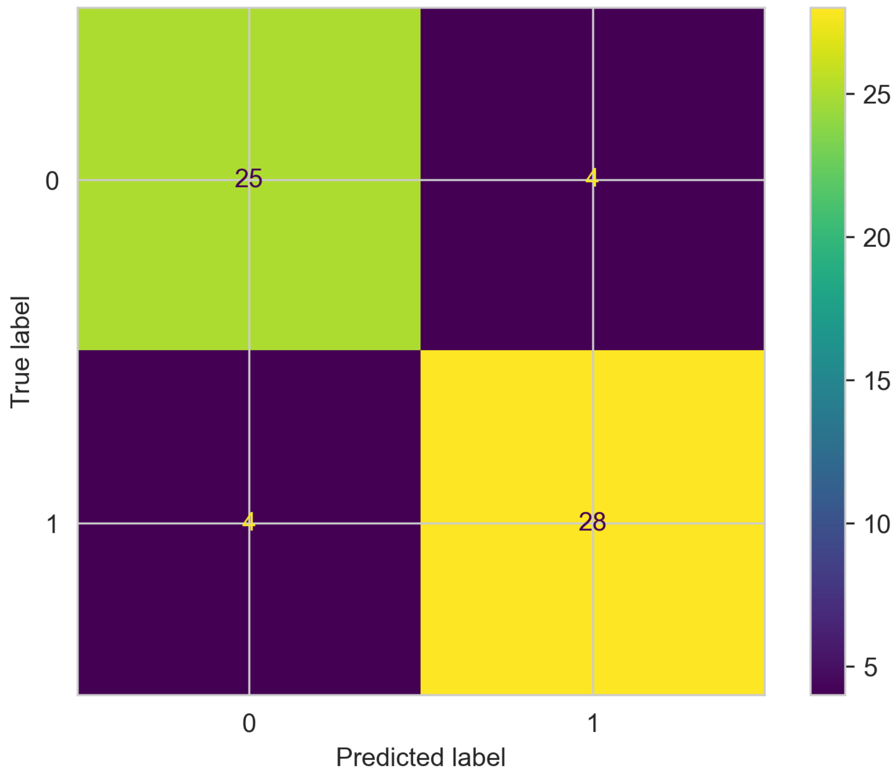
4.2 Test/Evaluation experimental procedure and analysis of results

We conducted a test of the model and obtained the following results:

4.2.1 Plots training and validation accuracy and loss for each fold on shared plots



4.2.2 Plot confusion matrix



4.2.3 Classification report

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

         0.0       0.86      0.86      0.86        29
         1.0       0.88      0.88      0.88        32

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

4.2.4 Showing the first 20 items of model prediction results

```
pred = ANN_model.predict(X_test).reshape(-1)  # Reshape predictions to 1D array
pred_binary = (pred >= 0.5).astype(int)  # Convert probabilities to binary predictions (0 or 1)

print("True heart attack chances      :", y_test.astype(int)[:20])
print("Predicted heart attack chances :", pred_binary[:20])
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
True heart attack chances      : [0 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0 0 0]
Predicted heart attack chances : [0 0 1 0 1 1 1 0 0 1 1 0 1 0 1 1 1 0 0 0]
```

4.3 Discuss the main results and their quality

4.3.1 Discussion of Previous Results

- **Variability and Average Performance:** Accuracy ranges from 81.9% to 88.5%, with an average around 85%. Indicates dependence on training data, typical in real-world scenarios.
- **Highest Achieved Accuracy:** The high of 88.5% under favorable conditions suggests good model performance potential.

4.3.2 Quality of the Results

- **Consistency:** Model shows reasonable consistency with minor deviations in fold performances, suggesting appropriate architecture and training.

4.3.3 Recommendations for Improvement

- **Hyperparameter Optimization:** Experiment with layers, units, learning rates, and optimizer settings.
- **Enhanced Regularization:** Introduce dropout or increase regularization parameters.

- **Feature Engineering:** Consider additional features or transformations to improve performance.
- **Increased Data:** More data or data augmentation techniques could enhance learning and generalization.
- **Advanced Model Architectures:** Explore more sophisticated architectures for potentially better results.

Chapter 5

Conclusions

5.1 Conclusions

The machine learning project aimed at predicting heart attack occurrences using Artificial Neural Networks (ANN) demonstrated promising outcomes. Key conclusions from the project include:

1. **Model Effectiveness**: The ANN model achieved a best accuracy rate of 88.5%, with performance variability across different folds indicating typical real-world scenario challenges.

2. **Robustness and Generalization**: The use of K-fold cross-validation ensured the model's robustness and ability to generalize well over unseen data, supported by the systematic approach to training and evaluation.

3. **Efficiency**: Early stopping during training prevented overfitting and enhanced computational efficiency, showcasing the model's capability to perform under optimal and limited resource scenarios.

5.2 Recommendations for Future Work

Based on the project's outcomes, the following recommendations are proposed to enhance future work:

1. **Hyperparameter Tuning**: Further exploration of hyperparameters such as the number of layers, neurons, and learning rates could potentially improve the model's accuracy and efficiency.

2. **Advanced Architectures**: Investigating more complex network architectures like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) may provide better insights and predictions due to their advanced pattern recognition capabilities.

3. **Data Augmentation**: Increasing the dataset size or implementing data augmentation strategies could help the model learn more comprehensive patterns, improving both accuracy and generalization.

4. **Feature Engineering**: Additional feature engineering might uncover more subtle patterns in the data that could significantly impact model performance.

5. **Regularization Techniques**: Integrating advanced regularization techniques like dropout could reduce overfitting further and enhance the model's prediction on new, unseen data.

6. **Cross-disciplinary Approaches**: Combining insights from fields such as epidemiology with machine learning could refine predictions and lead to more personalized healthcare interventions.

These enhancements could drive forward the development of more sophisticated models, ultimately contributing to the early detection and prevention of heart attacks.

Chapter 6

References

Bibliography

[1] A. Géron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems," 3rd ed., O'Reilly Media, Inc., 2021, ch. 10, "Introduction to Artificial Neural Networks with Keras."

[2] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into Deep Learning," available online: Dive into Deep Learning.

[3] "Flowchart for Artificial Neural Network (ANN)," ResearchGate, 2020, [Online]. Available: Flowchart for ANN.

[4] "Mathematics of artificial neural networks," Wikipedia, [Online]. Available: Mathematics of ANNs.

[5] XenonStack, "Artificial Neural Network Applications," 2023, available online: ANN Applications.

[6] P. Singh, "First Neural Network for Beginners Explained with Code," Towards Data Science, July 15, 2020, [Online]. Available: First Neural Network.

[7] MLTut, "Implementation of Artificial Neural Network in Python," available online: Implementation of ANN in Python.

[8] D. F. Campos, "Neural Network Cross Validation and Visualization," Kaggle, 2023, available online: Neural Network Cross Validation.

[9] neoglez, "K-Fold and ANN," GitHub, 2022, [Online]. Available: K-Fold and ANN on GitHub.

[10] S. Gunjal, "Tutorial: K-Fold Cross-Validation," Kaggle, 2023, [Online]. Available: K-Fold Cross-Validation.

[11] S. MechLearn, "Deep Tutorial 1: ANN and Classification," Kaggle, 2023, [Online]. Available: Deep Tutorial 1.

[12] Stack Exchange, "K-Fold Cross-Validation," Baeldung on CS, 2023, [Online]. Available: K-Fold Cross-Validation Article.

[13] "Pseudo-code of the back-propagation algorithm in training ANN," ResearchGate, 2018, [Online]. Available: Pseudo-code of back-propagation.

[14] "Optimization Methods for Large-Scale Machine Learning," Semantic Scholar, 2017, [Online]. Available: Optimization Methods.

[15] "What is the time complexity for training a neural network using back-propagation?," Artificial Intelligence Stack Exchange, 2019, [Online]. Available: Time Complexity for Training ANN.

Appendix A

Project Source Codes

To facilitate further learning and replication of our project's results, we have made the codebase available online. This allows educators, students, and other researchers to view, download, and modify the source code according to their needs.

A.1 Overview of the Code Repository

The code for this project is hosted on GitHub, a popular platform for version control and collaboration. It includes detailed documentation on how to set up the environment, run the models, and interpret the results.

A.2 How to Access the Code

The project code can be accessed through the following link:

Project Source Codes Link on GitHub

Please ensure you have Git installed on your system to clone the repository. You can clone the project using the following command in your terminal:

```
#!/bin/bash
# Script to clone the project repository
git clone https://github.com/Islam-hady9/Heart-Attack-
Analysis-Prediction-using-ANN.git
```

A.3 Navigating the Repository

The repository is structured as follows:

- **/docs**: Documentation files and additional resources
- **/src**: Complete source code for this project
- **/data**: Dataset used in this project
- **/results**: Output results and graphs for reference
- **/presentations**: Presentation for this project

A.4 Support and Contact

Should you encounter any issues or have questions regarding the project code, please open an issue on the GitHub repository, or contact us directly at eslamabdo71239@gmail.com.

A.5 Source Code

```
# Heart Attack Analysis & Prediction using ANN

# ---------------------------------------------------

# Introduction
# "Heart attacks" are a serious health issue worldwide. This
    analysis aims to find out which factors are connected to heart
    attacks and which ones affect it the most. By using data
    analysis and machine learning, the goal is to build a machine-
    learning model that can accurately predict the likelihood of
    someone having a heart attack. This can help people know if
    they are at risk and take steps to stay healthy and avoid heart
     attacks.

# ---------------------------------------------------
```

```python

# 1. Import Libraries

# EDA Libraries
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import plotly.subplots as sp
import plotly.graph_objects as go
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("whitegrid")

# Data Preprocessing Libraries
from datasist.structdata import detect_outliers
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_validate, KFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder, RobustScaler, LabelEncoder
from category_encoders import BinaryEncoder
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE

# Machine Learing and Deep Learning Libraries
import tensorflow as tf
from sklearn.feature_selection import SequentialFeatureSelector, SelectKBest, f_regression, RFE, SelectFromModel
from imblearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, f1_score, classification_report, roc_curve, roc_auc_score
from tensorflow.keras.utils import plot_model

# Ignore all warnings
import warnings
warnings.filterwarnings("ignore")

# -----------------------------------------------------

# 2. Data Exploration

df = pd.read_csv(r"Dataset\heart.csv")
df.sample(10)

# check the dataset shape
print("Number of Columns in data",df.shape[1])
print("---------------------------------------")
print("Number of Rows in data",df.shape[0])

# data information
df.info()

# checking for duplicated values
```

```
df.duplicated().sum()

# Removing duplicated data
df.drop_duplicates(inplace=True)

# checking if duplicated value has been removed
df.duplicated().sum()

# checking count the number of unique values in each column of the data
df.nunique()

# Descriptive analysis for numerical data
df.describe().style.background_gradient()

# ----------------------------------------------------

# 3. Exploratory Data Analysis

# 3.1. Univariate Analysis

# Exploration: Categorical Features

fig, axes = plt.subplots(3, 3, figsize=(13, 9))

# Creating a list of categorical features
cat_features = ['sex', 'cp', 'fbs', 'restecg', 'exng', 'slp', 'caa', 'thall', 'output']

#Looping through the subplots and create countplots for each feature
for i, ax in enumerate(axes.flat):
    if i < len(cat_features):
        sns.countplot(data=df, x=cat_features[i], ax=ax, palette="mako", orient='h')
        ax.set_title(f'Countplot for {cat_features[i]}', fontsize=14)

# Adjusting the layout for better visualization
plt.tight_layout()
plt.show()

# Exploration: Numerical Features

fig, axes = plt.subplots(1, 5, figsize=(15, 4))

# Creating a list of categorical features
cont_features = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']

#Looping through the subplots and create countplots for each feature
for i, ax in enumerate(axes.flat):
    if i < len(cont_features):
        sns.boxplot(data=df, x=cont_features[i], ax=ax, palette="mako", orient='h')
        ax.set_title(f'Boxplot for {cont_features[i]}', fontsize=16)
```

```python

# Adjusting the layout for better visualization
plt.tight_layout()
plt.show()

# Skewed Continuous Features Exploration

cont_columns = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']
fig, axes = plt.subplots(ncols=len(cont_columns), figsize=(18, 5))

# Plot distribution plots for each skewed column
for i, column in enumerate(cont_columns):
    sns.histplot(data=df, x=column, kde=True, ax=axes[i], color='
    skyblue')
    axes[i].set_title(f'Distribution of {column}', fontsize=14)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

plt.tight_layout()
plt.show()

# 3.2. Bivariate Analysis

#The Effect of Age on Risk of Heart Attack (Output)

# Creating a histogram using Plotly Express to visualize the
    relationship between age and the risk of heart attack
fig = px.histogram(df, x='age', color='output', title='The Effect
    of Age on Risk of Heart Attack (Output)',
                   labels={'age': 'Age', 'output': 'Output'},
                   marginal='box', barmode='group',
                   color_discrete_sequence=['#48a890', '#234457'],
    text_auto=True
                   )

# Customizing the layout of the histogram
fig.update_layout(
    xaxis=dict(tickmode='linear', dtick=2),  # Adjusting x-axis
    tick settings
    bargap=0.1  # Setting the gap between bars
)

# Customizing gridlines on the plot
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridcolor='lightgray')

# Customizing the background colors
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()

# The Effect of Sex on Risk of Heart Attack (Output)

# Filtering the DataFrame to separate male and female data
df_male = df[df['sex'] == 1]
df_female = df[df['sex'] == 0]

```

```
# Counting the occurrences of heart attack presence (output) for males and females
male_counts = df_male['output'].value_counts()
female_counts = df_female['output'].value_counts()

colors = ['#234457', '#48a890']

# Creating subplots for male and female distributions
fig = make_subplots(rows=1, cols=2, subplot_titles=('Male', 'Female'), specs=[[{'type':'domain'}, {'type':'domain'}]])

# Adding a pie chart for male heart attack presence
fig.add_trace(go.Pie(values=male_counts, name='Male',
                     marker=dict(colors=colors)), 1, 1)

# Adding a pie chart for female heart attack presence
fig.add_trace(go.Pie(values=female_counts, name='Female',
                     marker=dict(colors=colors)), 1, 2)

# Customizing the hole in the pie charts
fig.update_traces(hole=.4)

# Customizing the overall layout, title, and annotations
fig.update_layout(title_text='The Effect of Sex on Risk of Heart Attack (Output)', title_font=dict(size=18), title_x=0.5, title_y=0.95,
                 annotations=[dict(text='Male', x=0.22, y=0.45, font_size=25, showarrow=False),
                 dict(text='Female', x=0.78, y=0.45, font_size=25, showarrow=False)])

# Customizing background colors
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()

# The Effect of Sex on Risk of Heart Attack (Output)

# Creating a histogram to visualize the distribution of chest pain types (cp) with respect to heart attack risk (output)
fig = px.histogram(df, x='cp', color='output', title='The Effect of Cp (Chest Pain Types) on Risk of Heart Attack (Output)',
                   labels={'cp': 'Chest Pain Types', 'output': 'Output'}, barmode='group',
                   color_discrete_sequence=['#48a890', '#234457'], text_auto=True
                   )

# Customizing the gap between bars in the histogram
fig.update_layout(
    bargap=0.1
)

# Customizing the x-axis to show tick values and labels for different chest pain types
fig.update_xaxes(showgrid=True, gridcolor='lightgray', tickvals=[0, 1, 2, 3],
```

```
                 ticktext=['Typical Angina (0)', 'Atypical Angina (1)', 'Non-Anginal Pain (2)', 'Asymptomatic (3)'])

# Customizing the appearance of the y-axis
fig.update_yaxes(showgrid=True, gridcolor='lightgray')

# Customizing background colors
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()

# The Effect of Resting ECG Results (restecg) on Risk of Heart Attack (Output)

# Creating a histogram to visualize the effect of Resting ECG Results (restecg) on Heart Attack Risk (output)
fig = px.histogram(df, x='restecg', color='output',
                   title='The Effect of Resting ECG Results (restecg) on Risk of Heart Attack (Output)',
                   labels={'restecg': 'Resting ECG Results (restecg)', 'output': 'Output'}, barmode='group',
                   color_discrete_sequence=['#48a890', '#234457'],
                   category_orders={'restecg': ['0', '1', '2']}, text_auto=True
                   )

# Customizing the x-axis tick values and labels
fig.update_xaxes(tickvals=[0, 1, 2], ticktext=['Normal (0)', 'ST-T Wave Abnormality (1)', 'Probable/Definite LVH (2)'])

# Customizing the background color and gridlines
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()

# The Effect of Exercise-Induced Angina (exng) on Risk of Heart Attack (Output)

# Creating a histogram to visualize the relationship between Exercise-Induced Angina (exng) and the risk of heart attack (Output)
fig = px.histogram(df, x='exng', color='output', title='Exercise-Induced Angina (exng) vs. Risk of Heart Attack (Output)',
                   labels={'exng': 'Exercise-Induced Angina (exng)', 'output': 'Output'},
                   barmode='group',
                   color_discrete_sequence=['#48a890', '#234457'], text_auto=True
                   )

# Customizing layout: adjusting the gap between bars, marker appearance, gridlines, and title
fig.update_layout(
    bargap=0.1
)
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
```

```
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()

# The Effect of Oldpeak on Risk of Heart Attack (Output)

# Creating a histogram to visualize the relationship between Oldpeak and the risk of heart attack (Output)
fig = px.histogram(df, x='oldpeak', color='output', title='The Effect of Oldpeak on Risk of Heart Attack (Output)',
                   labels={'oldpeak': 'Oldpeak', 'output': 'Output'}, barmode='group',
                   color_discrete_sequence=['#48a890', '#234457'], text_auto=True )

# Customizing layout: adjusting the gap between bars, marker appearance, gridlines, and title
fig.update_layout(
    bargap=0.1
)
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()

# The Effect of Slope of ST Segment (slp) on Risk of Heart Attack (Output)

# Creating a histogram to visualize the relationship between the Slope of ST Segment (slp) and the risk of heart attack (Output)
fig = px.histogram(df, x='slp', color='output', title='The Effect of Slope of ST Segment (slp) on Risk of Heart Attack (Output)',
                   labels={'slp': 'Slope of ST Segment', 'output': 'Output'}, barmode='group',
                   color_discrete_sequence=['#48a890', '#234457'], text_auto=True )

# Customizing layout: adjusting the gap between bars, marker appearance, gridlines, and title
fig.update_layout(
    bargap=0.1
)
fig.update_xaxes(showgrid=True, gridcolor='lightgray', tickvals=[0, 1, 2, 3],
                 ticktext=['Downsloping (0)', 'Flat (1)', 'Upsloping (2)'])
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()

# The Effect of Number of Major Vessels Colored by Fluoroscopy (CAA) on Risk of Heart Attack (Output)

# Creating a histogram to visualize the relationship between the Number of Major Vessels (caa) and the risk of heart attack (Output)
fig = px.histogram(df, x='caa', color='output', barmode='group',
```

```
                   title='The Effect of Number of Major Vessels Colored by Fluoroscopy (CAA) on Risk of Heart Attack (Output)',
                   color_discrete_sequence=['#48a890', '#234457'],
                   labels={'caa': 'CAA (Number of Major Vessels)', 'output': 'Output'}, text_auto=True )

# Customizing layout: adjusting the title, font size, and background color
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.show()

# The Effect of Thalassemia Type (Thall) and Risk of Heart Attack (Output)

# Creating a histogram to visualize the relationship between Thalassemia Type (Thall) and the risk of heart attack (Output)
fig = px.histogram(df, x='thall', color='output', title='The Effect of Thalassemia Type (Thall) and Risk of Heart Attack (Output)',
                   labels={'thall': 'Thalassemia Type', 'output': 'Output'}, barmode='group',
                   color_discrete_sequence=['#48a890', '#234457'], text_auto=True )

# Customizing layout: adjusting the gap between bars, marker appearance, gridlines, and title
fig.update_layout(
    bargap=0.1
)
fig.update_xaxes(showgrid=True, gridcolor='lightgray', tickvals=[0, 1, 2, 3],
                 ticktext=['None (Normal) (0)', 'Fixed Defect (1)', 'Reversible Defect (2)', 'Thalassemia (3)'])
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.show()


# 3.3. Multivariate Analysis

# The Effect of Resting Blood Pressure (trtbps) and Age on Heart Attack Risk

# Creating a scatter plot
fig = px.scatter(df, x='age', y='trtbps', color=df['output'].astype(str),
                 title='The Effect of Resting Blood Pressure (trtbps) and Age on Heart Attack Risk ',
                 labels={'age': 'Age', 'trtbps': 'Resting Blood Pressure'},
                 color_discrete_sequence=['#48a890', '#234457'])

# Customizing the background color and gridlines
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
```

```
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.update_layout(legend_title_text='Output') # Rename the legend
fig.show()

# The Effect of Serum Cholesterol Levels (chol) and Age on Heart Attack Risk

# Creating a scatter plot
fig = px.scatter(df, x='age', y='chol', color=df['output'].astype(str),
                 title='The Effect of Serum Cholesterol Levels (chol) and Age on Heart Attack Risk',
                 labels={'age': 'Age', 'chol': 'Serum Cholesterol Levels'},
                 color_discrete_sequence=['#48a890', '#234457'])

# Customizing the background color and gridlines
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.update_layout(legend_title_text='Output')
fig.show()

# Maximum Heart Rate During Exercise (thalachh) and Age on Heart Attack Risk

# Creating a scatter plot
fig = px.scatter(df, x='age', y='thalachh', color=df['output'].astype(str),
                 title='The Effect of Maximum Heart Rate During Exercise (thalachh) and Age on Heart Attack Risk',
                 labels={'age': 'Age', 'thalachh': 'Maximum Heart Rate During Exercise'},
                 color_discrete_sequence=['#48a890', '#234457'])

# Customizing the background color and gridlines
fig.update_layout(plot_bgcolor='white', paper_bgcolor='white')
fig.update_xaxes(showgrid=True, gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridcolor='lightgray')
fig.update_layout(legend_title_text='Output')
fig.show()

# -----------------------------------------------------

# 4. Data Preprocessing

# 4.1. Handling Missing Data

# checking for missing values in data
df.isna().sum()

# 4.2. Handling Categorical Data

# Working with Nominal Features with pandas `get_dummies` function.
df = pd.get_dummies(df, columns=['cp', 'fbs', 'restecg', 'exng', 'slp', 'caa', 'thall'])

```

```python
encoded = list(df.columns)
print("{} total features after one-hot encoding.".format(len(
    encoded)))

df.head()

# 4.3. Handling Outliers

numerical_features = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak
    ']

# Detect outliers in numerical features
outliers_indices = detect_outliers(df, features=numerical_features,
     n=0)
number_of_outliers = len(outliers_indices)

print(f'Number of outliers in the Data: {number_of_outliers}')

# 4.4. Check The Distribution of Classes

# Assuming your DataFrame is named "df"
plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
sns.countplot(x='output', data=df)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Distribution of Class')
plt.show()

# Check the distribution of classes
print("Class distribution in dataset:")
print(df['output'].value_counts())


# 4.5. Data Split to Train and Test Sets

# First we extract the x Featues and y Label
X = df.drop(['output'], axis=1)
y = df['output']

X.shape, y.shape

# Then we Split the data into training and testing sets (80%
    training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state
    =42,
                                                   )

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))


# 4.6. Feature Scaling

# Robust Scaling Continuous Features with RobustScaler
```

```
numerical_features = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak
    ']

# Creating a RobustScaler instance
scaler = RobustScaler()

# Transforming (scaling) the continuous features in the training
    and testing data
X_train_cont_scaled = scaler.fit_transform(X_train[
    numerical_features])
X_test_cont_scaled = scaler.transform(X_test[numerical_features])

# Replacing the scaled continuous features in the original data
X_train[numerical_features] = X_train_cont_scaled
X_test[numerical_features] = X_test_cont_scaled

# Display the modified X_train with scaled features
display(X_train)

# -----------------------------------------------------

# 5. ANN Model Training with Cross-Validation and Evaluation

# 5.1. Build ANN Model Training with Cross-Validation

# The data had (boolean) values. Converting everything into (np.
    float32).
X_train = np.asarray(X_train).astype(np.float32)
y_train = np.asarray(y_train).astype(np.float32)

X_test = np.asarray(X_test).astype(np.float32)
y_test = np.asarray(y_test).astype(np.float32)

# ANN Model

# Set the random seed for reproducibility
tf.random.set_seed(42)

# Define the number of folds for KFold cross-validation
n_splits = 5
kfold = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# Prepare to collect scores and histories
accuracies = []
all_histories = []

# KFold Cross Validation
for train_index, val_index in kfold.split(X_train):
    # Split data
    X_train_kfold, X_val_kfold = X_train[train_index], X_train[
    val_index]
    y_train_kfold, y_val_kfold = y_train[train_index], y_train[
    val_index]

    # Create a new instance of the model (to reinitialize weights)
    ANN_model = tf.keras.Sequential([
```

```
        tf.keras.layers.Dense(32, activation="relu", input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(16, activation="relu"),
        tf.keras.layers.Dense(1, activation="sigmoid")
    ])

    # Compile the model
    ANN_model.compile(loss="binary_crossentropy",
                  optimizer=tf.keras.optimizers.Adam(),
                  metrics=["accuracy"])

    # Early stopping callback
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    )

    # Fit the model
    history = ANN_model.fit(X_train_kfold, y_train_kfold,
                        epochs=100,
                        validation_data=(X_val_kfold, y_val_kfold),
                        callbacks=[early_stopping],
                        verbose=0)  # Set verbose to 0 to reduce output

    # Collect the history from each fold
    all_histories.append(history)

    # Evaluate the model on the validation set
    scores = ANN_model.evaluate(X_val_kfold, y_val_kfold, verbose=0)
    accuracies.append(scores[1])  # Assume that the accuracy is the second metric

# Print the accuracy for each fold
print("Accuracy for each fold:", accuracies)

# Print the average accuracy
print("Average accuracy:", np.mean(accuracies))


# ## This code performs K-Fold cross-validation on an artificial neural network (ANN) using TensorFlow. Let's break down each part:
#
# ### Setting the Random Seed
# ```python
# tf.random.set_seed(42)
# ```
# This sets the random seed to 42 for TensorFlow, ensuring reproducibility of results. When you set a seed, the sequence of random numbers generated will be the same every time you run the code.
#
# ### Defining K-Fold Cross-Validation
# ```python
```

````python
# n_splits = 5
# kfold = KFold(n_splits=n_splits, shuffle=True, random_state=42)
# ```
# K-Fold cross-validation splits the dataset into `n_splits` (here, 5) folds. The `shuffle=True` parameter shuffles the data before splitting it into folds, and `random_state=42` ensures the shuffling is reproducible.
#
# ### Preparing to Collect Scores and Histories
# ```python
# accuracies = []
# all_histories = []
# ```
# These lists will store the accuracy scores and training histories for each fold.
#
# ### K-Fold Cross-Validation Loop
# ```python
# for train_index, val_index in kfold.split(X_train):
# ```
# This loop iterates over each fold, splitting the data into training and validation sets for each fold.
#
# ### Splitting Data for Each Fold
# ```python
# X_train_kfold, X_val_kfold = X_train[train_index], X_train[val_index]
# y_train_kfold, y_val_kfold = y_train[train_index], y_train[val_index]
# ```
# Here, `X_train` and `y_train` are split into training and validation sets based on the indices provided by `kfold.split`.
#
# ### Creating the Model
# ```python
# ANN_model = tf.keras.Sequential([
#     tf.keras.layers.Dense(32, activation="relu", input_shape=(X_train.shape[1],)),
#     tf.keras.layers.Dense(16, activation="relu"),
#     tf.keras.layers.Dense(1, activation="sigmoid")
# ])
# ```
# A new instance of the neural network is created for each fold. This network has three layers:
# 1. An input layer with 32 neurons and ReLU activation.
# 2. A hidden layer with 16 neurons and ReLU activation.
# 3. An output layer with 1 neuron and sigmoid activation (suitable for binary classification).
#
# ### Compiling the Model
# ```python
# ANN_model.compile(loss="binary_crossentropy",
#                   optimizer=tf.keras.optimizers.Adam(),
#                   metrics=["accuracy"])
# ```
# The model is compiled with:
````

```
# - Binary cross-entropy loss (appropriate for binary classification).
# - Adam optimizer.
# - Accuracy as the metric to evaluate performance.
#
# ### Early Stopping Callback
# ```python
# early_stopping = tf.keras.callbacks.EarlyStopping(
#     monitor='val_loss',
#     patience=10,
#     restore_best_weights=True
# )
# ```
# Early stopping monitors the validation loss and stops training if it doesn't improve for 10 epochs. It also restores the best weights to prevent overfitting.
#
# ### Fitting the Model
# ```python
# history = ANN_model.fit(X_train_kfold, y_train_kfold,
#                         epochs=100,
#                         validation_data=(X_val_kfold, y_val_kfold),
#                         callbacks=[early_stopping],
#                         verbose=0)
# ```
# The model is trained for up to 100 epochs, using the training and validation sets for the current fold. Early stopping is applied, and the training history is recorded.
#
# ### Collecting Histories and Evaluating the Model
# ```python
# all_histories.append(history)
# scores = ANN_model.evaluate(X_val_kfold, y_val_kfold, verbose=0)
# accuracies.append(scores[1])
# ```
# The training history is stored, and the model is evaluated on the validation set. The accuracy score is saved.
#
# ### Printing Results
# ```python
# print("Accuracy for each fold:", accuracies)
# print("Average accuracy:", np.mean(accuracies))
# ```
# Finally, the accuracy for each fold and the average accuracy across all folds are printed.
#
# ### Summary
# This code performs K-Fold cross-validation to evaluate the performance of an ANN on a given dataset. It ensures the model's results are reproducible, uses early stopping to prevent overfitting, and calculates the accuracy for each fold as well as the average accuracy across all folds.

ANN_model.summary()

# Visualize the model
```

```python
plot_model(ANN_model, to_file='model_plot.png', show_shapes=True,
    show_layer_names=True)

# 5.2. ANN Model Evaluation

# Evaluate model on the test dataset
loss, accuracy = ANN_model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')

# Plots training and validation accuracy and loss for each fold on
    shared plots, making it easy to compare performance across
    folds.

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))
fig.suptitle('Training and Validation Metrics Across Folds')

for i, history in enumerate(all_histories):
    ax1.plot(history.history['accuracy'], label=f'Fold {i+1} Train
    Accuracy')
    ax1.plot(history.history['val_accuracy'], label=f'Fold {i+1}
    Val Accuracy', linestyle='--')
    ax2.plot(history.history['loss'], label=f'Fold {i+1} Train Loss
    ')
    ax2.plot(history.history['val_loss'], label=f'Fold {i+1} Val
    Loss', linestyle='--')

ax1.set_ylabel('Accuracy')
ax1.legend()
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend()

# save the figure
plt.savefig('folds_plot.png', dpi=300, bbox_inches='tight')
plt.show()

# Predictions on test data
y_pred = ANN_model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_binary)
print("\nConfusion Matrix:\n", cm)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(cm)
disp.plot()

# save the figure
plt.savefig('cm_plot.png', dpi=300, bbox_inches='tight')
plt.show()

# Display classification report
print("\nClassification Report:\n", classification_report(y_test,
    y_pred_binary))
```

```
pred = ANN_model.predict(X_test).reshape(-1)  # Reshape predictions to 1D array
pred_binary = (pred >= 0.5).astype(int)  # Convert probabilities to binary predictions (0 or 1)

print("True heart attack chances      :", y_test.astype(int)[:20])
print("Predicted heart attack chances :", pred_binary[:20])
```

Appendix B

Project Team Plan and Achievement

The project team for the "Heart Attack Analysis and Prediction using Artificial Neural Network" comprised three members: Islam Abd-Elhady Hassanein Mohamed, Enas Ragab Abdel-Latif Mohamed, and Mariam Tarek Saad Mohamed. The collective expertise in data science and machine learning formed the core strength of the team, enabling a focused and effective approach to tackling the project's challenges.

B.1 Team Plan

1. **Roles and Responsibilities:**

   - **Islam Abd-Elhady Hassanein Mohamed** focused on data preprocessing, including data cleaning and normalization, ensuring that the dataset was ready for modeling.
   - **Enas Ragab Abdel-Latif Mohamed** took the lead on model development, specifically working

on designing and implementing the neural network architecture.

- **Mariam Tarek Saad Mohamed** was responsible for testing and validation, which involved running the model, analyzing the output, and tuning the model parameters based on the performance metrics.

2. **Schedule and Milestones:** The project was planned over a three-month period, with the first month dedicated to data exploration and preprocessing, the second month to model building and initial testing, and the third month to final testing, validation, and documentation.

3. **Collaboration and Communication:** Weekly meetings were held to discuss progress, challenges, and next steps. Regular updates were shared among team members via a shared online platform, which also housed all project documentation and code for easy access.

B.2 Achievements

1. **Model Development:** Successfully implemented an Artificial Neural Network that could predict heart attack occurrences with an accuracy of up to 88.5%. The model was robust, benefiting from K-fold cross-validation to ensure its effectiveness on unseen data.

2. **Innovative Solutions:** Applied several state-of-the-art machine learning techniques, such as early stop-

ping and Adam optimization, to improve model training efficiency and effectiveness.

3. **Knowledge Sharing:** The team maintained a comprehensive log of their findings and methodologies in a shared document that was later converted into a detailed project report, contributing to the academic and practical understanding of applying neural networks in medical prediction.

4. **Community Contribution:** Presented the findings in a well-received session during the university's annual data science conference, highlighting the project's impact and potential applications in healthcare.

5. **Future Readiness:** Prepared a roadmap for future improvements, including potential enhancements in model architecture and the exploration of additional data sources to refine predictive accuracy.

This project not only achieved its goal of developing a predictive model but also fostered a collaborative and innovative environment that significantly enhanced the team's analytical and problem-solving skills. The successful completion of the project stands as a testament to the effectiveness of the planned approach and the dedication of each team member.

Appendix C

Link to the Presentation File

In this section, you will find the link to the presentation file. The presentation file provides an overview of our project and includes detailed information about our findings and recommendations. Please click on the following link to access the presentation file:

Presentation File Link