MLP Implementation

Multilayer-Perceptron Implementation On Heart Diseases Dataset

Author M.M.Mirzaie

June 7, 2025

Contents

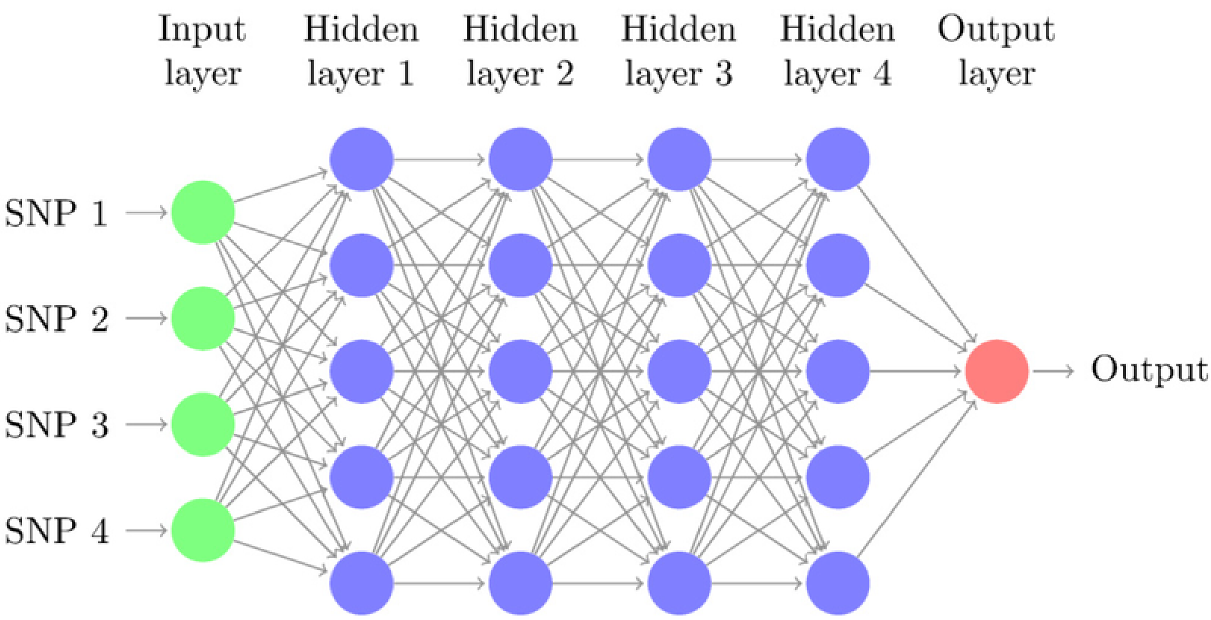
1 Introduction

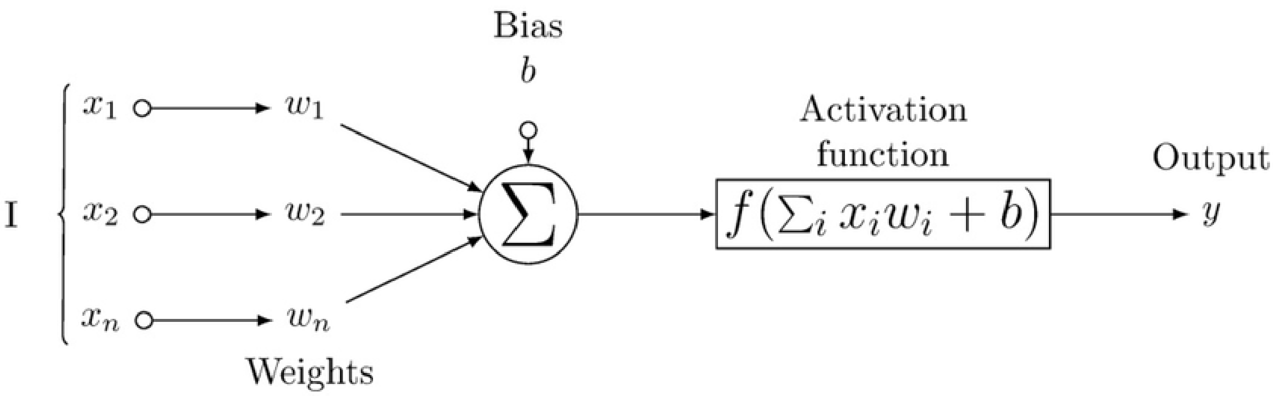
The Perceptron, that neural network whose name evokes how the future looked from the perspective of the 1950s, is a simple algorithm intended to perform binary classification; i.e. it predicts whether input belongs to a certain category of interest or not.

The perceptron is a linear classifier — an algorithm that classifies input by separating two categories with a straight line. Input is typically a feature vector xmultiplied by weights w and added to a bias b: y = w * x + b.

Perceptrons produce a single output based on several real-valued inputs by forming a linear combination using input weights (and sometimes passing the output through a non-linear activation function).

Rosenblatt built a single-layer perceptron ; it did not include multiple layers, which allow neural networks to model a feature hierarchy. It was, therefore, a shallow neural network, which ended up preventing his perceptron from performing non-linear classification, such as the classic logic XOR function (an XOR operator trigger when input exhibits either one trait or another, but not both; it stands for "exclusive OR").





2 Data Set

2.1 Dataset Properties

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The ”target” field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

2.2 Parameter Meaning

```
Only 14 attributes used:
1. #3  (age)
2. #4  (sex)
3. #9  (cp)
4. #10 (trestbps)
5. #12 (chol)
6. #16 (fbs)
7. #19 (restecg)
8. #32 (thalach)
9. #38 (exang)
10. #40 (oldpeak)
11. #41 (slope)
12. #44 (ca)
13. #51 (thal)
14. #58 (num)       (the predicted attribute)

Complete attribute documentation:
1 id: patient identification number
2 ccf: social security number (I replaced this with a dummy value of 0)
3 age: age in years
4 sex: sex (1 = male; 0 = female)
5 painloc: chest pain location (1 = substernal; 0 = otherwise)
6 painexer (1 = provoked by exertion; 0 = otherwise)
7 relrest (1 = relieved after rest; 0 = otherwise)
8 pncaden (sum of 5, 6, and 7)
9 cp: chest pain type
-- Value 1: typical angina
-- Value 2: atypical angina
-- Value 3: non-anginal pain
-- Value 4: asymptomatic
10 trestbps: resting blood pressure (in mm Hg on admission to the hospital)
11 htn
12 chol: serum cholestoral in mg/dl
13 smoke: I believe this is 1 = yes; 0 = no (is or is not a smoker)
14 cigs (cigarettes per day)
15 years (number of years as a smoker)
16 fbs: (fasting blood sugar > 120 mg/dl)  (1 = true; 0 = false)
```

3 Preamble Of ML Process

3.1 Importing Libraries

```python
# Importing the pandas library, which provides data structures and data analysis tools for Python.
import pandas as pd

# Importing the numpy library, which is a fundamental package for numerical computations in Python.
import numpy as np

# Importing TensorFlow, an open-source library for machine learning and deep learning tasks.
# TensorFlow provides tools to build and train neural networks and is widely used in the field of AI.
import tensorflow as tf

# Importing the load_iris function from sklearn.datasets, which is a utility to load the famous Iris dataset.
# The Iris dataset is a classic dataset often used for classification tasks in machine learning.
from sklearn.datasets import load_iris

# Importing train_test_split from sklearn.model_selection, which is a utility to split datasets into training and testing subsets.
from sklearn.model_selection import train_test_split

# Importing OneHotEncoder from sklearn.preprocessing, which is used to convert categorical variables into a format
# that can be provided to machine learning algorithms to improve predictions.
# This technique is often used for encoding categorical features.
from sklearn.preprocessing import OneHotEncoder

# Importing StandardScaler from sklearn.preprocessing, which is used to standardize features by removing the mean
# and scaling to unit variance. This technique is crucial for many machine learning algorithms, as it helps
# ensure that all features contribute equally to the distance calculations.
from sklearn.preprocessing import StandardScaler

```

3.2 Read the CSV dataset

```
#reading heart disease classification dataset. Please refer this documentation: /
  content/data/documentation.pdf
df = '/content/heart.csv'
df = pd.read_csv(df)
df

```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 13 columns

3.3 Preliminary Statistical Analysis

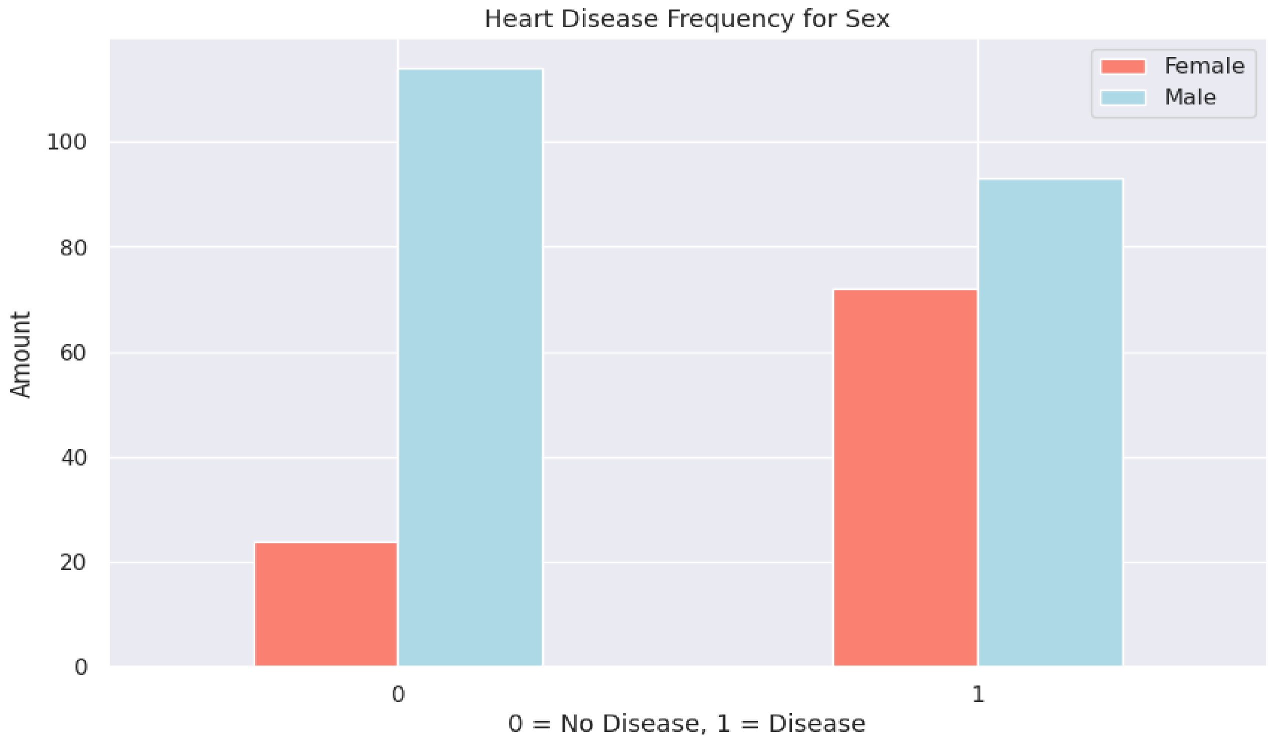
```
df.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
df.describe()

```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

```
# Create a plot
pd.crosstab(df.output, df.sex).plot(kind="bar",
figsize=(10,6),
color=["salmon", "lightblue"]);

# Add some attributes to it
plt.title("Heart Disease Frequency for Sex")
plt.xlabel("0 = No Disease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female", "Male"])
plt.xticks(rotation=0); # keep the labels on the x-axis vertical

```



```
    #extracting input and output data from the training dataset
    #excluding all column, except a last one
    X = df.iloc[:, :-1]
    #excluding only the last column
    y = df.iloc[:, -1]
    X

```

```
y

```

	output
0	1
1	1
2	1
3	1
4	1
...	...
298	0
299	0
300	0
301	0
302	0

303 rows × 1 columns

```python
#1190 samples and 11 features
print(X.shape)
#binary targets
print(y.shape)

```

```
(303, 13)
(303,)
```

It Means X have 303 rows and 13 columns(features) and y have have 303 row and just one column that is the output(target)

3.4 Splitting dataset

```python
#splitting dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=24, stratify=y)

```

train_test_split is used to split a dataset into training and testing subsets.

This is a common step in machine learning to evaluate the performance of a model on unseen data.

X: The feature matrix (input data).

y: The target vector (output labels).

test_size=0.2: Specifies the proportion of the dataset to include in the test split. Here, 0.2 means 20% of the data will be used for testing, and the remaining 80% will be used for training.

random_state=24: Ensures reproducibility of the split by setting a seed for the random number generator. Using the same random_state value will produce the same split every time the code is run.

stratify=y: Ensures that the class distribution in the target variable y is preserved in both the training and testing sets.

This is particularly useful for imbalanced datasets, where one class might dominate the others.

3.5 Scaling The Futures

```
#scaling the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

There are different methods for scaling data in this tutorial we will use a method called standardization.

The standardization method uses this formula:

z = (x - u) / s

Where z is the new value, x is the original value, u is the mean and s is the standard deviation.

X_train = scaler.fit_transform(X_train):

fit Computes the mean and standard deviation of each feature in the training data (X_train).

transform: Applies the standardization (z-score normalization) to the training data.

The training data is scaled based on its own mean and standard deviation.

X_test = scaler.transform(X_test):

transform: Applies the same scaling (computed from X_train) to the test data (X_test).

This ensures that the test data is scaled consistently with the training data.

4 Defines a Multi-Layer Perceptron (MLP) model

```
    #input layer: 11 input features
    #hidden layer1: 16 nodes (using ReLU activation function)
    #output layer: 2 classes (using sigmoid because it's a classification problem)

    #defining the MLP model
    mlp_cl = tf.keras.Sequential([
    tf.keras.layers.Dense(16,activation='relu',input_shape=(13,),name="Input layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden layer"),
    tf.keras.layers.Dense(1, activation='sigmoid', name="Output layer")
    ])
])

```

Input Layer:

tf.keras.layers.Dense(16, activation='relu', input_shape=(13,)): A dense (fully connected) layer with 16 nodes.

activation='relu': Uses the ReLU (Rectified Linear Unit) activation function, which outputs. This introduces non-linearity.

input_shape=(13,): Specifies the input shape, which is 13 features in this case.

Hidden Layer:

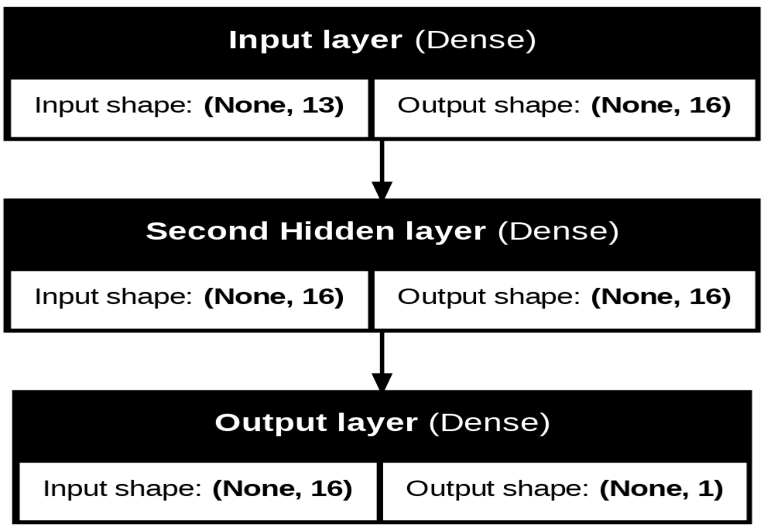
tf.keras.layers.Dense(16, activation='relu'): Another dense layer with 16 nodes and ReLU activation. No need to specify input_shape here, as it is inferred from the previous layer.

Output Layer:

tf.keras.layers.Dense(1, activation='sigmoid'): A dense layer with 1 node (output). activation='sigmoid': Outputs a value between 0 and 1, suitable for binary classification problems (e.g., two classes).

4.1 Visualization

```
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.utils import plot_model
plot_model(mlp_cl, to_file='model_plot.png', show_shapes=True, show_layer_names=
  True)

```



5 Compile And Train The Model

5.1 Compile The Model

```
mlp_cl.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])

```

The compile() method configures the model for training. It is a crucial step in the model-building process, where you specify how the model should be trained.

optimizer='adam':

This argument specifies the optimization algorithm to be used during training. Here, the Adam optimizer is selected. Adam is a popular choice because it combines the advantages of two other extensions of stochastic gradient descent (SGD): AdaGrad and RMSProp. It adapts the learning rate for each parameter, which helps in faster convergence and better performance.

loss='binary_crossentropy':

This argument defines the loss function that the model will optimize during training. In this case, the loss function is binary cross-entropy, which is commonly used for binary classification tasks (where there are two classes, e.g., 0 and 1). The binary cross-entropy loss measures the performance of a model whose output is a probability value between 0 and 1. It quantifies how well the predicted probabilities match the actual binary labels.

metrics=['accuracy']:

This argument specifies the evaluation metric(s) to monitor during training and evaluation. Here, accuracy is chosen as the metric. Accuracy measures the proportion of correct predictions made by the model. By including it in the compilation step, the model will report accuracy during training and validation phases.

5.2 Train The Model

```
#training the model
mlp_cl.fit(X_train, y_train, epochs=100, validation_split=0.2)

```

The fit() method is used to train the model on the provided training data. It adjusts the model's parameters (weights) based on the input data and the corresponding labels (targets) to minimize the specified loss function.

epochs=100:

This parameter specifies the number of epochs to train the model. An epoch is one complete pass through the entire training dataset. In this case, the model will be trained for 100 epochs. The model will learn patterns in the training data over these iterations.

validation_split=0.2:

This parameter specifies the fraction of the training data to be used as validation data. Here, 0.2 means that 20% of the training data will be set aside for validation. The validation data is used to evaluate the model's performance during training, allowing you to monitor overfitting. The model's performance on the validation set is reported at the end of each epoch.

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8406 - loss: 0.3304 - val_accuracy: 0.7959 - val_loss: 0.4373
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8143 - loss: 0.3957 - val_accuracy: 0.7551 - val_loss: 0.4867
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8307 - loss: 0.3551 - val_accuracy: 0.7551 - val_loss: 0.4877
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8419 - loss: 0.3409 - val_accuracy: 0.7959 - val_loss: 0.4382
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8409 - loss: 0.3197 - val_accuracy: 0.7551 - val_loss: 0.6213
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8023 - loss: 0.4060 - val_accuracy: 0.7347 - val_loss: 0.5615
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8046 - loss: 0.3528 - val_accuracy: 0.7959 - val_loss: 0.4619
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8195 - loss: 0.3710 - val_accuracy: 0.7959 - val_loss: 0.4705
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.8229 - loss: 0.3710 - val_accuracy: 0.7959 - val_loss: 0.4722
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.8458 - loss: 0.3154 - val_accuracy: 0.7347 - val_loss: 0.5123
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8356 - loss: 0.3501 - val_accuracy: 0.7347 - val_loss: 0.5439
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8298 - loss: 0.3359 - val_accuracy: 0.7347 - val_loss: 0.5380
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8623 - loss: 0.3110 - val_accuracy: 0.7347 - val_loss: 0.5194
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8153 - loss: 0.3494 - val_accuracy: 0.7347 - val_loss: 0.5371
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8711 - loss: 0.3120 - val_accuracy: 0.7755 - val_loss: 0.4384
```

Figure 1: 100 Epoch Process

```
Epoch 1/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.7019 - loss: 0.5965 - val_accuracy: 0.7959 - val_loss: 0.4747
Epoch 2/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6978 - loss: 0.6315 - val_accuracy: 0.5714 - val_loss: 0.6726
Epoch 3/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6960 - loss: 0.6148 - val_accuracy: 0.6939 - val_loss: 0.5264
Epoch 4/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7604 - loss: 0.5261 - val_accuracy: 0.7551 - val_loss: 0.4826
Epoch 5/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7230 - loss: 0.5442 - val_accuracy: 0.6327 - val_loss: 0.5447
Epoch 6/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7121 - loss: 0.5491 - val_accuracy: 0.7551 - val_loss: 0.4957
Epoch 7/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6818 - loss: 0.5771 - val_accuracy: 0.6735 - val_loss: 0.5109
Epoch 8/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.7055 - loss: 0.5401 - val_accuracy: 0.7551 - val_loss: 0.4941
Epoch 9/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7132 - loss: 0.5542 - val_accuracy: 0.6327 - val_loss: 0.5904
Epoch 10/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7187 - loss: 0.5304 - val_accuracy: 0.7755 - val_loss: 0.4790
Epoch 11/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7170 - loss: 0.4993 - val_accuracy: 0.5714 - val_loss: 0.6341
Epoch 12/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7285 - loss: 0.5612 - val_accuracy: 0.6735 - val_loss: 0.5461
Epoch 13/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7408 - loss: 0.5169 - val_accuracy: 0.6939 - val_loss: 0.5298
Epoch 14/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.6901 - loss: 0.5882 - val_accuracy: 0.7143 - val_loss: 0.4997
Epoch 15/300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.7308 - loss: 0.5383 - val_accuracy: 0.6939 - val_loss: 0.5387
Epoch 16/300
```

Figure 2: 300 Epoch Process(The Accuracy will increases But Not Necessary)

6 Evaluation And Prediction

6.1 Evaluating The Model

```python
#After training, evaluate the mlp_cl model on the test set (X_test, y_test) to see  how well the model is performing)
loss, accuracy = mlp_cl.evaluate(X_test, y_test)
print(f"test accuracy: {accuracy:.4f}")

```

The evaluate() method is used to assess the model's performance on a given dataset. In this case, it evaluates the model on the test set, which consists of X_test (features) and y_test (true labels). This method computes the loss value and any additional metrics specified during the model compilation (in this case, accuracy).

Method returns two values: the loss and the accuracy of the model on the test set. These values are unpacked into the variables loss and accuracy.

print(f"test accuracy: accuracy:.4f"):

This line prints the test accuracy to the console. The f-string format allows for embedding expressions inside string literals. The :.4f format specifier means that the accuracy will be printed as a floating-point number with four decimal places.

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.8282 - loss: 0.4702
test accuracy: 0.8361
```

Figure 3: Accuracy with 100 Epoch Process

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.8391 - loss: 0.4124
test accuracy: 0.8525
```

Figure 4: Accuracy with 300 Epoch Process

6.2 Making Predictions

```python
#making predictions
predictions = mlp_cl.predict(X_test)
predicted_classes = (predictions > 0.5).astype(int)
print("Predicted classes", predicted_classes.flatten())


print("True Output", list(y.head(50)))

```

The predict() method is used to generate predictions from the trained model for the input data X_test.

predictions:

This variable stores the output probabilities from the predict() method. Each value in predictions represents the predicted probability that the corresponding input sample belongs to the positive class.

predicted_classes = (predictions ¿ 0.5).astype(int):

In this line, the predicted probabilities are converted to binary class labels. The expression (predictions ¿ 0.5) creates a boolean array where each entry is True if the corresponding predicted probability is greater than 0.5 (indicating class 1) and False otherwise (indicating class 0).

The .astype(int) method then converts the boolean values to integers, resulting in an array of predicted class labels (0 or 1).

print("Predicted classes", predicted_classes.flatten()):

Finally, this line prints the predicted class labels to the console. The flatten() method is used to convert the array into a 1D array (if it is multidimensional), making it easier to read. The output will display the predicted classes for all samples in the test set.

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
Predicted classes [1 1 1 0 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 0
 0 1 1 0 1 0 1 1 1 1 0 1 1 0 0 1 1 0 0 1 1 1 1 1]
True Output [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Figure 5: Prediction with 100 Epoch Process

6.3 Accuracy Plots

Creates a matplotlib figure (12x4 inches) with a subplot (1 row, 2 columns, first position). Plots training and validation accuracy from history.history (from model.fit) over epochs. Sets title to "Model Accuracy," labels axes, and adds a legend to distinguish the two lines.

```
    import matplotlib.pyplot as plt
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()

```

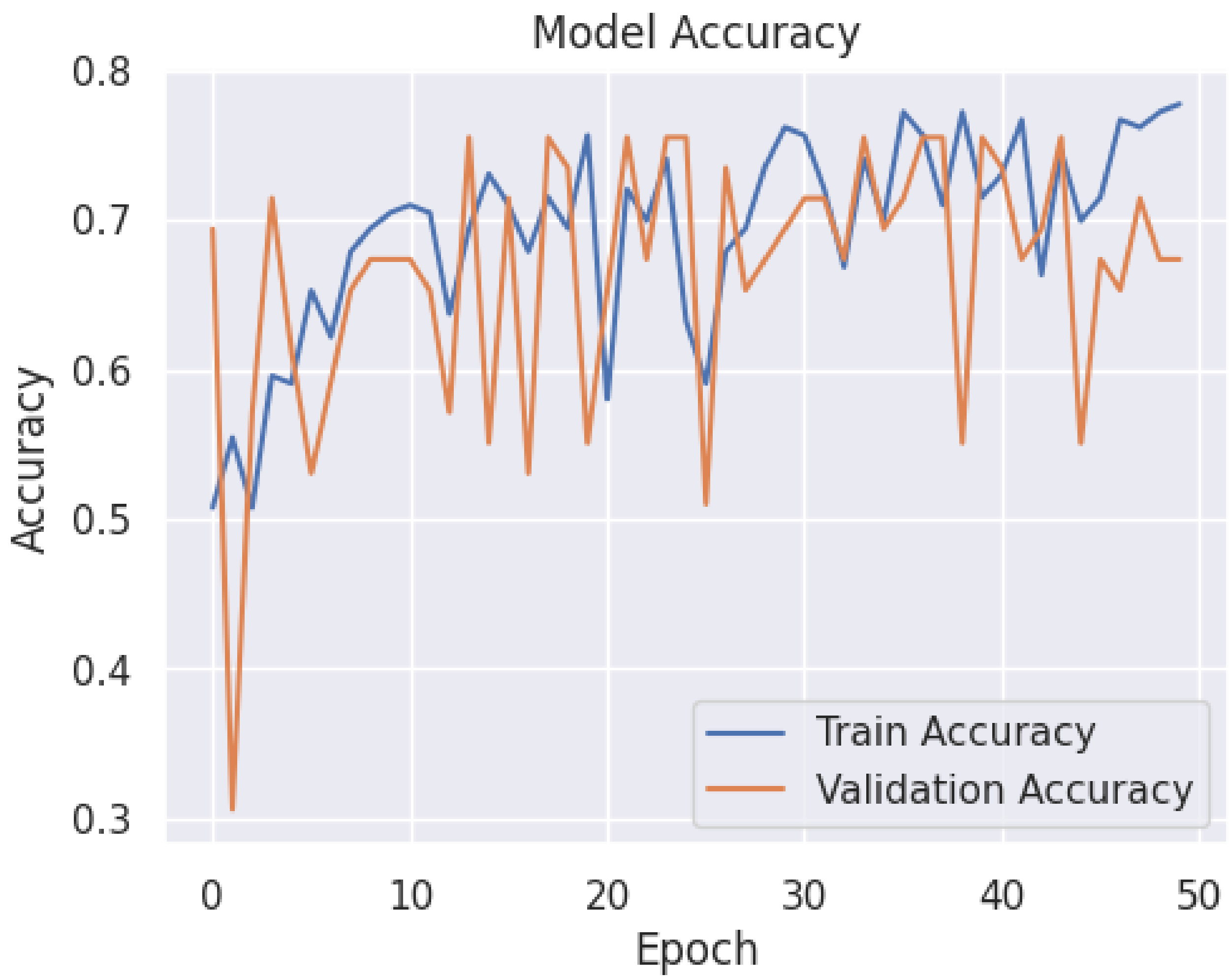


Figure 6:

6.4 Confusion Matrix

```
import seaborn as sns # seaborn gets shortened to sns
corr_matrix = df.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix,
annot=True,
linewidths=0.5,
fmt= ".2f",
cmap="YlGnBu");

```

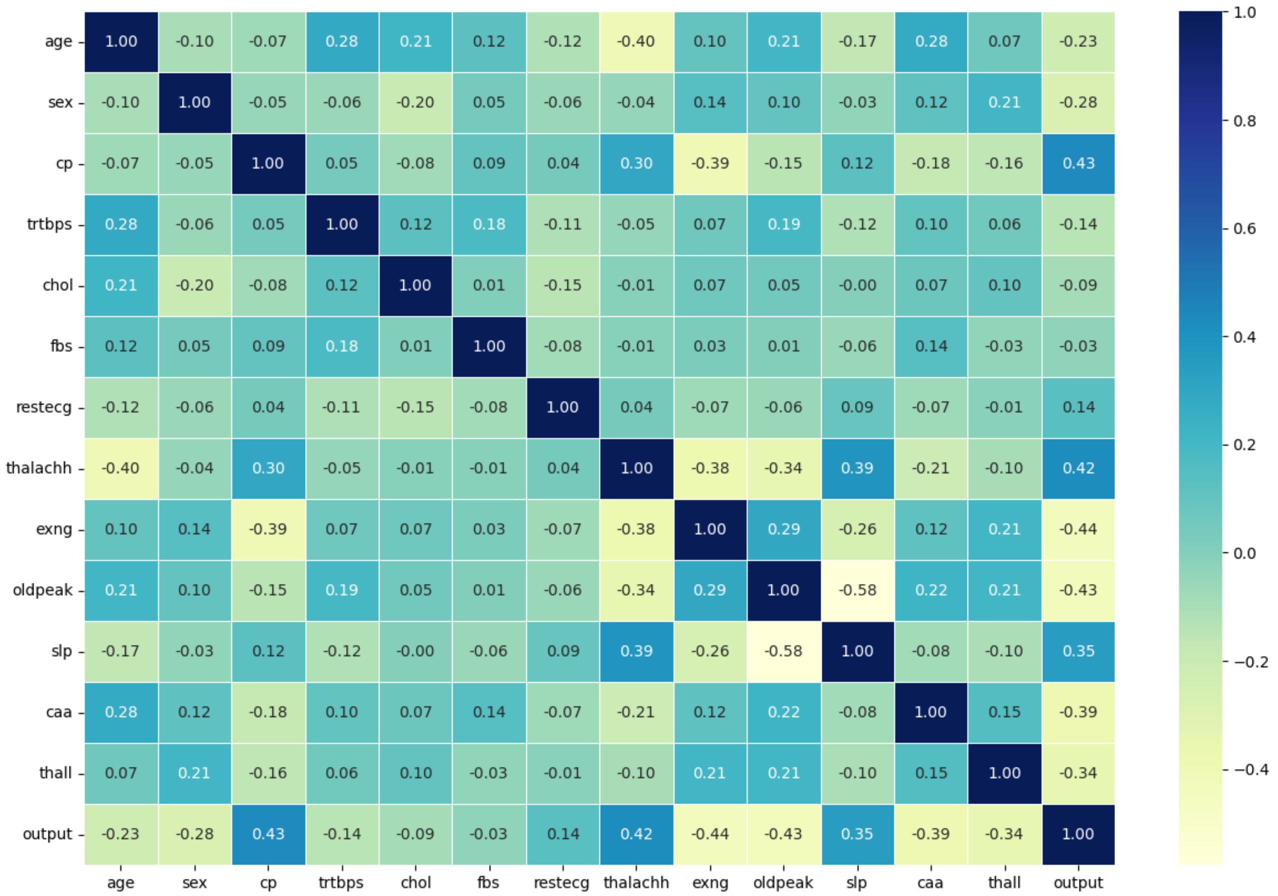


Figure 7:

6.5 Scatter Plots

```
# Create another figure
plt.figure(figsize=(10, 6))

# Scatter with positive examples
plt.scatter(df.age[df.output==1],
df.thalachh[df.output==1],
c="salmon")

# Scatter with negative examples
plt.scatter(df.age[df.output==0],
df.thalachh[df.output==0],
c="lightblue")

# Add some helpful info
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease", "No Disease"]);

```

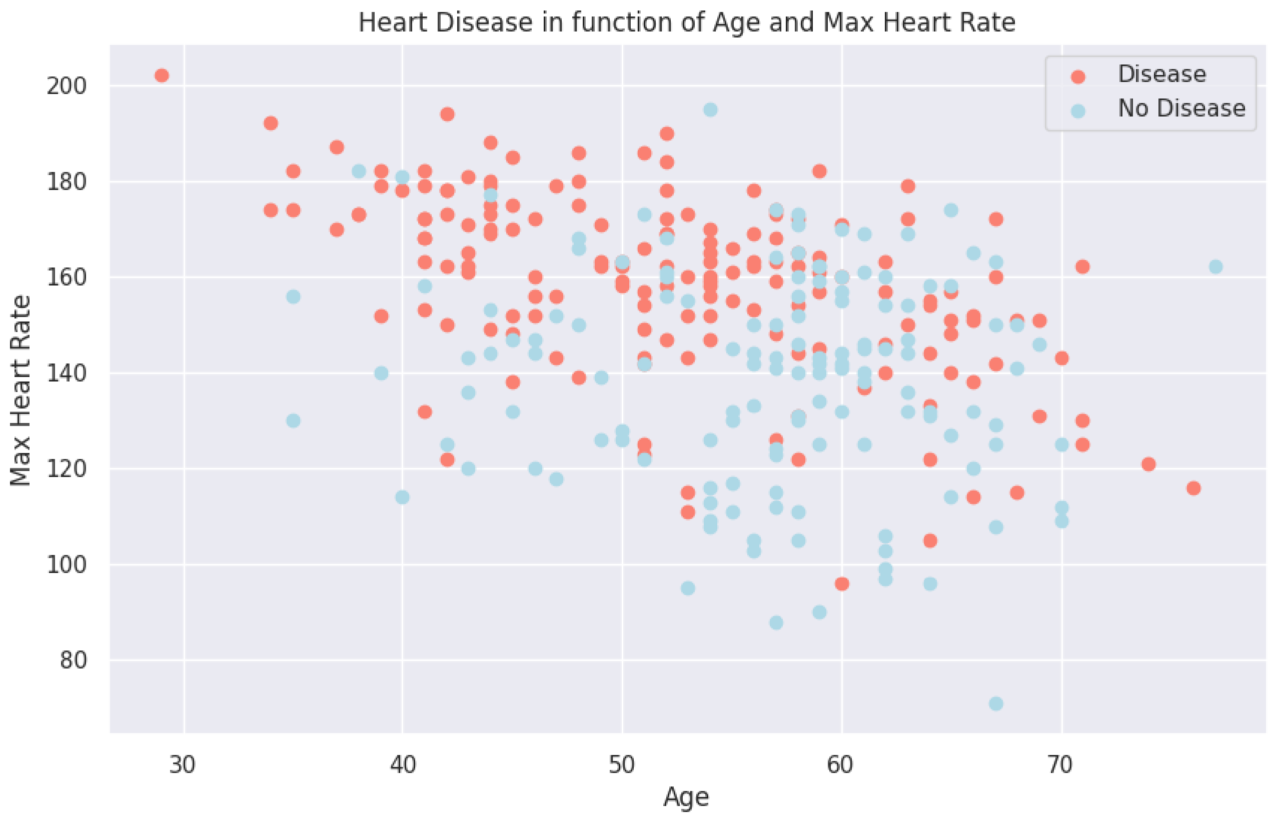


Figure 8:

7 Refrences

- www.youtube.com/@gcdatkin
- www.youtube.com/@pyhind
- https://medium.com
- https://grok.com