UFAZ L3 AI FINAL PROJECT REPORT

Imran Ibrahimli
L3 CS2
imran.ibrahimli@ufaz.az

Sanan Najafov
L3 CS2
sanan.najafov@ufaz.az

Sabina Hajimuradova
L3 CS1
sabina.hajimuradova@ufaz.az

1 Introduction

The project covers 2 machine learning algorithms: multilayer perceptron (MLP) and decision tree. The methods are applied to a dataset containing samples of medical measurements and the persons' heart condition (healthy or not), which is a binary classification problem. The aim of the project is to train both of the classifiers on the dataset, and compare the results using multiple evaluation metrics.

All of the code and figures are on github: `https://github.com/iibrahimli/ai_final_project`.

2 Dataset

The dataset we use in this project is the Cleveland Heart Disease dataset which can be downloaded from `https://archive.ics.uci.edu/ml/datasets/Heart+Disease`. The original dataset contains 303 samples and 75 attributes, though we use a reduced version that contains 14 attributes. Extensive descriptions and data types of the attributes can be found on the dataset webpage.

2.1 Exploratory Data Analysis

Initial analysis of the dataset provided to us shows that the attributes include both numerical and categorical features. The attributes and their type can be found in Table 1. The categorical attributes represent discrete states and therefore it is best to one-hot encode them in order to use as input to our neural network. One-hot encoding will create dummy columns and will increase the number of input features from 14 to 30. It is often helpful to compute and plot the so-called correlation matrix [1]. Higher (positive or negative) entries in the matrix indicate that the features are strongly (positively or negatively) correlated. Looking at the Figure 1, features that are most noticeably correlated with `target` are: `chest_pain_type`, `max_heart_rate_achieved`, and `st_slope`. There is also considerable negative correlation between `target` and `exercise_induced_angina`, `st_depression`.

Name	Type	Mean	Std	Min	Max
`age`	Numerical	54.366	9.082	29.000	77.000
`sex`	Categorical				
`chest_pain_type`	Categorical				
`resting_blood_pressure`	Numerical	131.623	17.538	94.000	200.000
`cholesterol`	Numerical	246.264	51.830	126.000	564.000
`fasting_blood_sugar`	Categorical				
`rest_ecg`	Categorical				
`max_heart_rate_achieved`	Numerical	149.646	22.905	71.000	202.000
`exercise_induced_angina`	Categorical				
`st_depression`	Numerical	1.039	1.161	0.000	6.200
`st_slope`	Categorical				
`num_major_vessels`	Categorical				
`thalassemia`	Categorical				
`target`	Categorical				

Table 1: Attribute types and descriptive statistics for numerical attributes

2.2 Answers to Questions

1. We want to predict the attribute `target`.
2. Binary classification, since there are 2 possible cases (healthy or not). Even though some algorithms (ex: neural networks) predict the probability of having a heart condition, which is a continuous variable, in the end the result has to be converted to a binary label through thresholding.
3. See Table 1 for types of attributes.
4. Using one-hot encoding, we can convert a discrete attribute that takes one of N values to a vector $\in \{0, 1\}^N$.
5. In the case of predicting the risk of heart disease, sensitivity would be more important than specificity, because the patients who have been falsely identified as healthy can neglect their heart condition, possibly leading to death. Thus it is better to have the minimal number of people whose heart condition goes undetected.

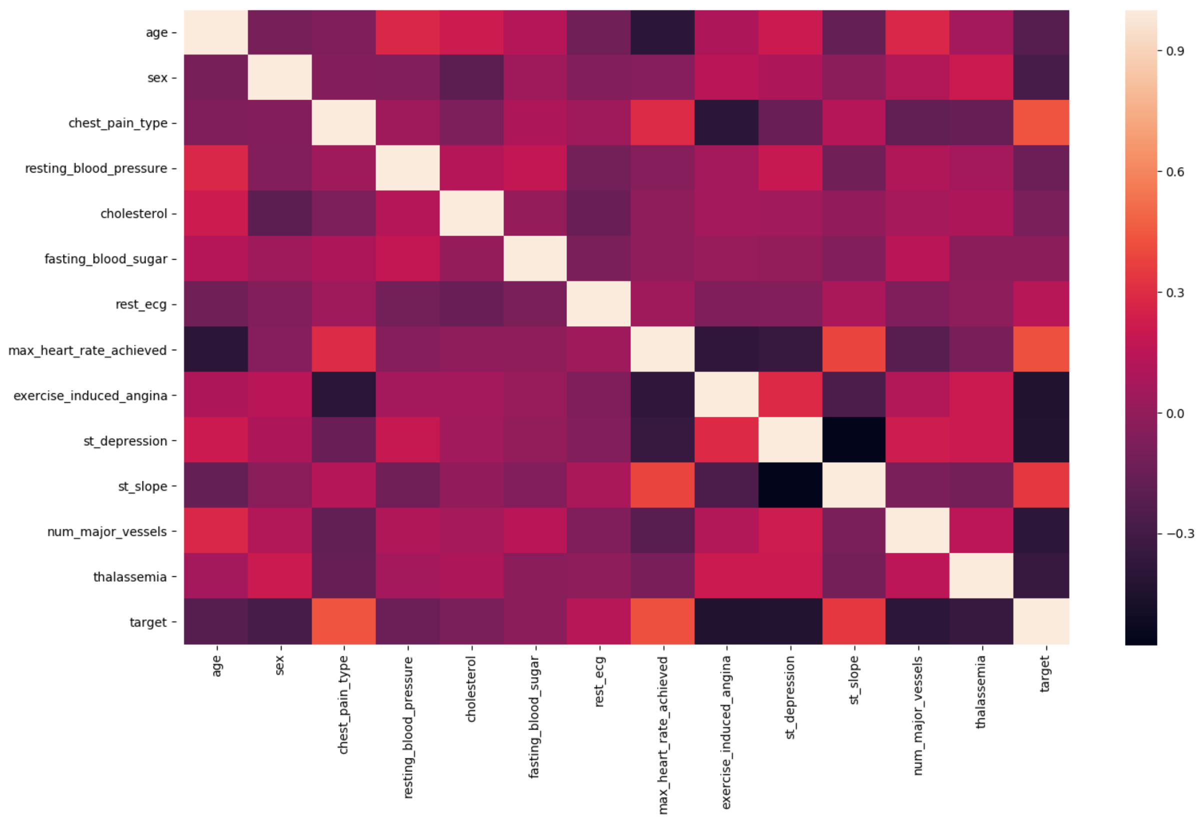


Figure 1: Correlation matrix for the attributes

3 Multilayer Perceptron (MLP)

The generic and well tested code from previous Practical Work was reused. The code required some easy modifications, such as the addition of binary cross-entropy loss, a few activation and metric functions. The network consists of 30 input neurons, 5 hidden neurons, and 1 output neuron (see Figure 2).

3.1 Answers to Questions

1. There are 2 weight matrices with shapes (30, 5) and (5, 1). The bias vectors have shapes (5,) and (1,). Mini-batches can be elegantly processed by defining the weight matrices appropriately (see 3.3).
2. We stop the training once the validation loss stops decreasing

3.2 Preprocessing

The data preprocessing for MLP consists of normalization and one-hot encoding. Categorical variables are one-hot encoded while numerical attributes are *column-wise* standardized using z-score. Z-score was chosen as the normalization method because it allows us to center the attributes around zero and limit their range while preserving the distribution of data. It is computed as

$$Z = \frac{X - \mu}{\sigma}$$

where X is the initial data vector, μ is the mean of X, and σ is the standard deviation of X. After normalization, the normalized data vector Z will have $\mu = 0$ and $\sigma = 1$. It is important to use the same mean and standard deviation for the normalization of training and test sets, hence, either the dataset must be split after normalization, or the mean and standard deviation of each column must be stored for later use. Normalization procedure is implemented in the function `feat_standardize()` in file `mlp/data.py`.

3.3 Training

Training is done through backpropagation introduced by Jeoff Hinton et al in [2]. The data (i.e. both input features and intermediate activation values) is represented as a matrix of samples, where each sample is a row vector. Therefore, the shape of the matrix is $n\_samples \times n\_features$. This simplifies training on mini-batches and derivation of backpropagation equations. The weights of layer l+1 W_l are represented as matrices of size $n_{l-1} \times n_l$ where n_{l-1} is the number of neurons in l-1$^{\text{th}}$ layer and n_l is the number of neurons in l$^{\text{th}}$ layer. For each layer, biases are stored as a vector $b_l \in \mathbb{R}^{n_l}$. Also, for each layer we store inputs to the activation function of that layer (i.e. the sum of inputs to neurons) z_l, and activation values a_l, both have the same size as b_l. This enables some optimizations during forward- and backpropagation. The weights are initialized using Xavier initialization [3] and the biases are initialized to zeros. Xavier initialization aims to stabilize the training from the start by sampling initial weights from $\mathcal{N}(0, \frac{1}{\sqrt{n_{l-1}}})$ which keeps the variance of layer activations the same throughout the layers.

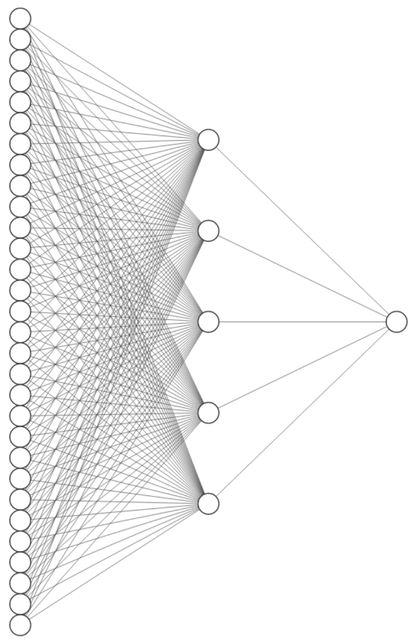

Input Layer ∈ $\mathbb{R}^{30}$
Hidden Layer ∈ $\mathbb{R}^{5}$
Output Layer ∈ $\mathbb{R}^{1}$

Figure 2: Network architecture

The relationship between layer l and $l - 1$ can be expressed as

$$Z_l = \sigma(A_{l-1}W_l + b_l)$$

and

$$A_l = \sigma(Z_l)$$

where σ denotes the activation function, capital A and Z denote matrix forms of a and z for mini-batches of inputs. The computation of partial derivatives under this formulation is easy and out of scope of this report. It is only worth mentioning that since we compute AW, knowing the partial derivative of a matrix product is enough to derive the gradient. For a matrix C such that $C = AB$ it can be shown that

$$\frac{\partial C}{\partial A} = B^T$$

and

$$\frac{\partial C}{\partial B} = A^T$$

The loss function J measures the error between ground truth label and predicted label. In this case, we use binary cross-entropy (BCE), because it is best suited for our problem. In our case, $J : \mathbb{R} \times \mathbb{R} \mapsto [0, \infty[$. The parameters are updated proportionally to the gradient of the loss function with respect to the parameter:

$$W_l = W_l - \alpha \frac{\partial J}{\partial W_l}$$

$$b_l = b_l - \alpha \frac{\partial J}{\partial b_l}$$

3.3.1 What Improved Performance

- *Xavier initialization*: As mentioned above, it accelerates the training and reduces the possibility of numerical issues.
- *Learning rate annealing*: Our MLP library supports learning rate annealing, so it is possible to pass a `tuple` (initial learning rate (lr), number of epochs to wait between lr multiplication, value to multiply lr by) instead of a `float`. We found out that for this dataset, (0.1, 100, 0.5) = [multiplying learning rate by 0.5 each 100 epochs, starting from learning rate of 0.1] worked well.
- *Larger mini-batch size*: Larger mini-batch size tends to smooth the gradient. We used a batch size of 32.
- *Early stopping*: Instead of letting the training run for a specified number of epochs, it is better to stop the training once validation loss does not decrease anymore. This may prevent overfitting.

3.4 Evaluation

To evaluate the network, we trained on 70% - 30% training-test split, and were able to achieve accuracy of 1 after some hyper-parameter tuning. This shows that the problem is easy for the neural network, as it was able to achieve such unrealistically high scores (Table 2). The evolution of loss (Figure 3), metrics (Figure 4), and screenshots of program run (Figure 5, Figure 6) are also included.

Metric	Score
Accuracy	1.0000
Precision	1.0000
Sensitivity	1.0000
Specificity	1.0000
F1	1.0000

Table 2: Test set scores (single run)

Metric	Score
Accuracy	0.9868
Precision	0.9649
Sensitivity	1.0000
Specificity	0.9794
F1	0.9819

Table 3: 5-fold CV scores

MLP with batch size=32, initial learning rate=0.1 (multiplied by 0.5 every 100 epochs)

Obviously, this result is not always reproducible, and highly depends on the split. It is better to use K-fold cross validation to evaluate the network (we use K=5). As seen in Table 3, the scores are still pretty high, though not exactly equal to 1. Sensitivity reached a score of 1, which is very important for our problem.

4 Decision Tree

The code from previous practical works was reused and some C++ code was converted to Python as well.

4.1 Preprocessing

Decision trees are not sensitive to data range, that is why numerical attributes are discretized into 3 groups instead of normalization. Categorical attributes are not one-hot encoded, since they are already discrete.

Potential problem: Categorical attributes may take more values than number of groups, which would result in some lost information. This could be fixed by modifying the code to be able to work with different number of groups for attributes.

4.2 Training

Training is exactly the same as in previous practical works, we use entropy to split data based on attributes that have the most discriminative power. Maximum tree depth is set to 4.

4.3 Evaluation

Training and test data was split with same ratio as in multilayer perceptron (80% - 20% since we consider the scores obtained with K=5 cross validation). The scores for test set were significantly lower than multilayer perceptron (Table 4, screenshot: Figure 7). This can stem from our implementation, which is not perfect (see Potential problem) or the fact that decision trees partition the feature space with orthogonal hyperplanes and it is difficult to do that for highly non-linear data without overfitting.

Metric	Score
Accuracy	0.8030
Precision	0.8750
Sensitivity	0.7780
Specificity	0.8400
F1	0.8240

Table 4: Decision tree test set scores

5 Conclusion

Neural network-based approach is superior in tackling this problem, as it scores higher than decision tree in every metric we consider (especially sensitivity).

References

[1] Nathaniel E. Helwig. Data, covariance, and correlation matrix. `http://users.stat.umn.edu/~helwig/notes/datamat-Notes.pdf`.

[2] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

6 Plots and Screnshots

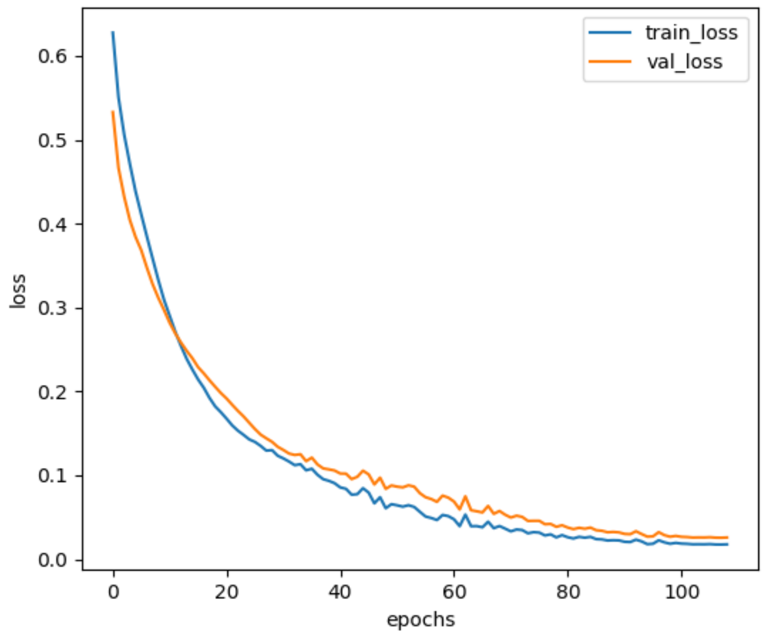


Figure 3: MLP loss evolution as training progresses

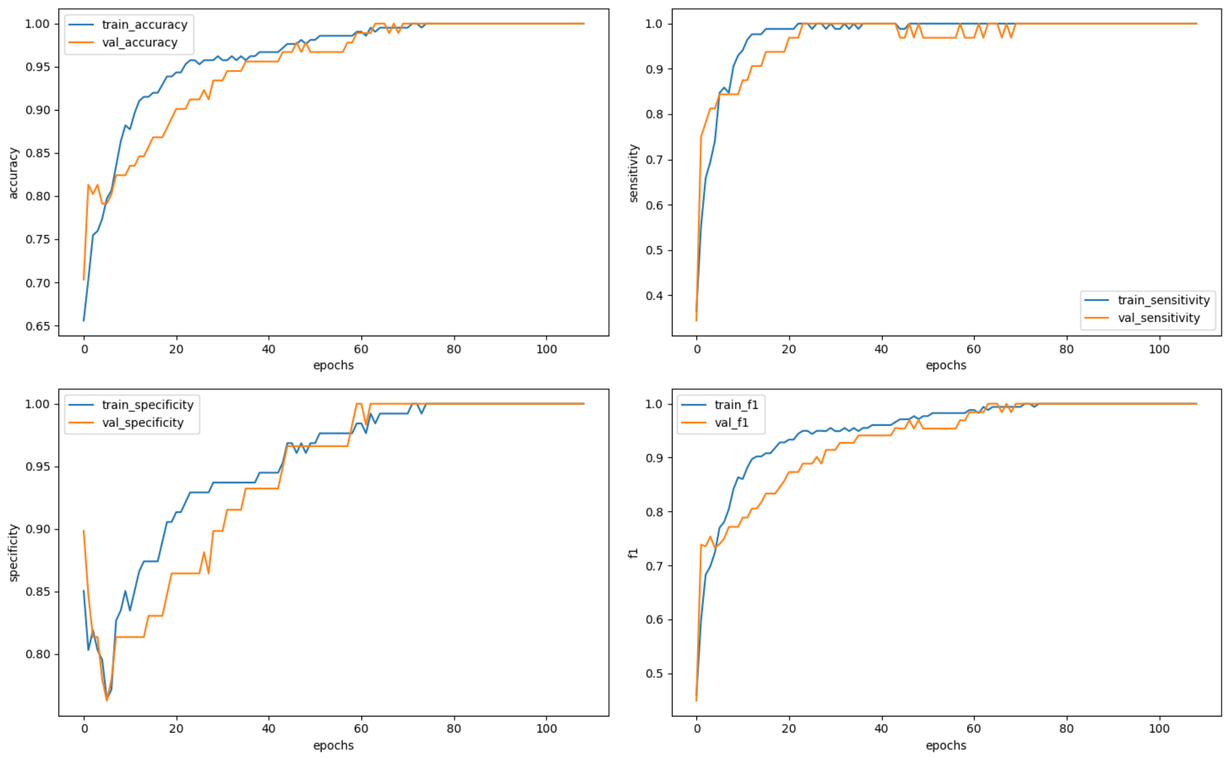
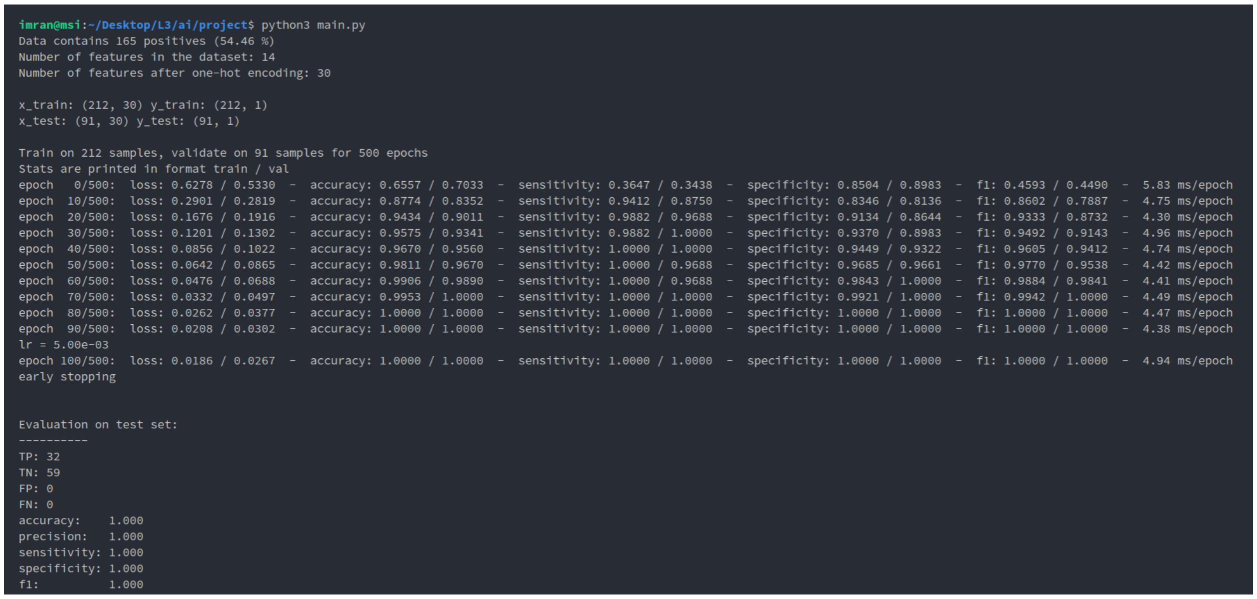


Figure 4: MLP metrics evolution as training progresses



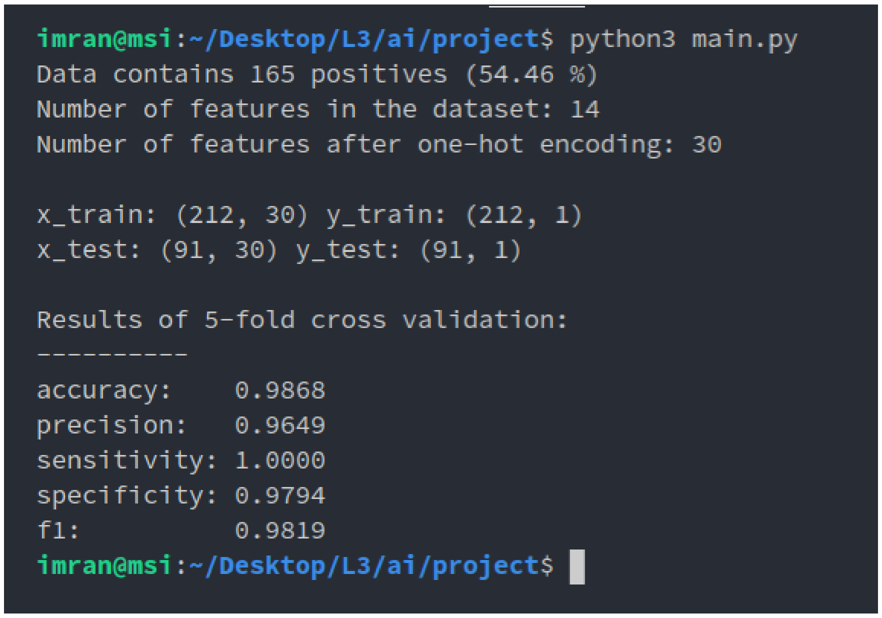
```
imran@msi:~/Desktop/L3/ai/project$ python3 main.py
Data contains 165 positives (54.46 %)
Number of features in the dataset: 14
Number of features after one-hot encoding: 30

x_train: (212, 30) y_train: (212, 1)
x_test: (91, 30) y_test: (91, 1)

Train on 212 samples, validate on 91 samples for 500 epochs
Stats are printed in format train / val
epoch   0/500:  loss: 0.6278 / 0.5330  -  accuracy: 0.6557 / 0.7033  -  sensitivity: 0.3647 / 0.3438  -  specificity: 0.8504 / 0.8983  -  f1: 0.4593 / 0.4490  -  5.83 ms/epoch
epoch  10/500:  loss: 0.2901 / 0.2819  -  accuracy: 0.8774 / 0.8352  -  sensitivity: 0.9412 / 0.8750  -  specificity: 0.8346 / 0.8136  -  f1: 0.8602 / 0.7887  -  4.75 ms/epoch
epoch  20/500:  loss: 0.1676 / 0.1916  -  accuracy: 0.9434 / 0.9011  -  sensitivity: 0.9882 / 0.9688  -  specificity: 0.9134 / 0.8644  -  f1: 0.9333 / 0.8732  -  4.30 ms/epoch
epoch  30/500:  loss: 0.1201 / 0.1302  -  accuracy: 0.9575 / 0.9341  -  sensitivity: 0.9882 / 1.0000  -  specificity: 0.9370 / 0.8983  -  f1: 0.9492 / 0.9143  -  4.96 ms/epoch
epoch  40/500:  loss: 0.0856 / 0.1022  -  accuracy: 0.9670 / 0.9560  -  sensitivity: 1.0000 / 1.0000  -  specificity: 0.9449 / 0.9322  -  f1: 0.9605 / 0.9412  -  4.74 ms/epoch
epoch  50/500:  loss: 0.0642 / 0.0865  -  accuracy: 0.9811 / 0.9670  -  sensitivity: 1.0000 / 0.9688  -  specificity: 0.9685 / 0.9661  -  f1: 0.9770 / 0.9538  -  4.42 ms/epoch
epoch  60/500:  loss: 0.0476 / 0.0688  -  accuracy: 0.9906 / 0.9890  -  sensitivity: 1.0000 / 0.9688  -  specificity: 0.9843 / 1.0000  -  f1: 0.9884 / 0.9841  -  4.41 ms/epoch
epoch  70/500:  loss: 0.0332 / 0.0497  -  accuracy: 0.9953 / 1.0000  -  sensitivity: 1.0000 / 1.0000  -  specificity: 0.9921 / 1.0000  -  f1: 0.9942 / 1.0000  -  4.49 ms/epoch
epoch  80/500:  loss: 0.0262 / 0.0377  -  accuracy: 1.0000 / 1.0000  -  sensitivity: 1.0000 / 1.0000  -  specificity: 1.0000 / 1.0000  -  f1: 1.0000 / 1.0000  -  4.47 ms/epoch
epoch  90/500:  loss: 0.0208 / 0.0302  -  accuracy: 1.0000 / 1.0000  -  sensitivity: 1.0000 / 1.0000  -  specificity: 1.0000 / 1.0000  -  f1: 1.0000 / 1.0000  -  4.38 ms/epoch
lr = 5.00e-03
epoch 100/500:  loss: 0.0186 / 0.0267  -  accuracy: 1.0000 / 1.0000  -  sensitivity: 1.0000 / 1.0000  -  specificity: 1.0000 / 1.0000  -  f1: 1.0000 / 1.0000  -  4.94 ms/epoch
early stopping


Evaluation on test set:
----------
TP: 32
TN: 59
FP: 0
FN: 0
accuracy:    1.000
precision:   1.000
sensitivity: 1.000
specificity: 1.000
f1:          1.000
```

Figure 5: MLP single run screenshot

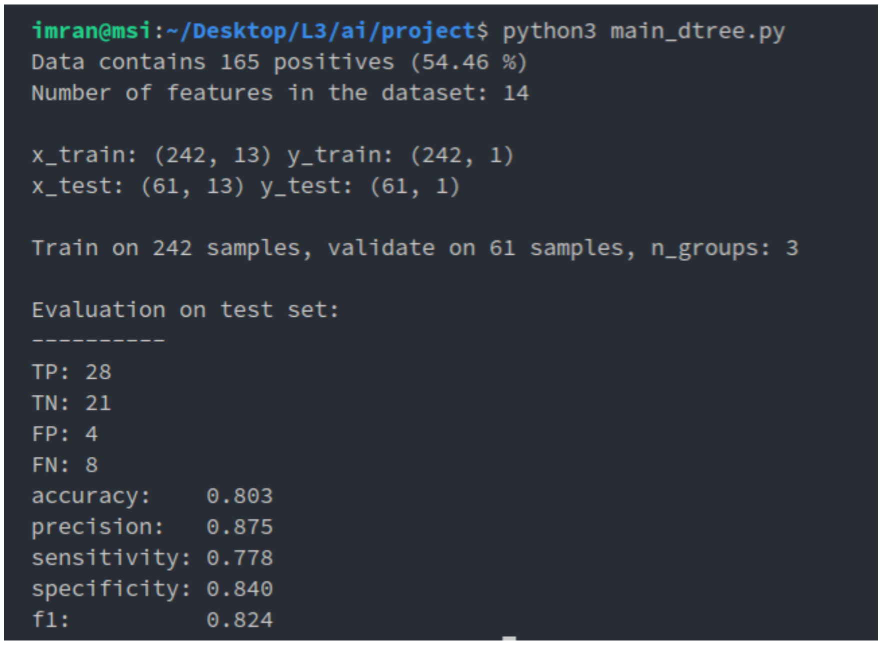


```
imran@msi:~/Desktop/L3/ai/project$ python3 main.py
Data contains 165 positives (54.46 %)
Number of features in the dataset: 14
Number of features after one-hot encoding: 30

x_train: (212, 30) y_train: (212, 1)
x_test: (91, 30) y_test: (91, 1)

Results of 5-fold cross validation:
----------
accuracy:    0.9868
precision:   0.9649
sensitivity: 1.0000
specificity: 0.9794
f1:          0.9819
imran@msi:~/Desktop/L3/ai/project$
```

Figure 6: MLP 5-fold CV run screenshot



```
imran@msi:~/Desktop/L3/ai/project$ python3 main_dtree.py
Data contains 165 positives (54.46 %)
Number of features in the dataset: 14

x_train: (242, 13) y_train: (242, 1)
x_test: (61, 13) y_test: (61, 1)

Train on 242 samples, validate on 61 samples, n_groups: 3

Evaluation on test set:
----------
TP: 28
TN: 21
FP: 4
FN: 8
accuracy:    0.803
precision:   0.875
sensitivity: 0.778
specificity: 0.840
f1:          0.824
```

Figure 7: Decision tree run screenshot