Lab : Predicting the risk of heart disease in patients

Aziz Zeynalli, Javidan Hajiyev

March 29, 2024

Abstract

His report explores the development and comparison of machine learning models for predicting heart disease risk in patients. The Cleveland Heart Disease Dataset, containing 14 physiological measurements for 303 patients, is used for training and evaluation. We implemented two models: a Multi-Layer Perceptron (MLP) with one hidden layer, and a Decision Tree with a maximum depth of 3. Both models were trained using one-hot encoded, normalized data with mini-batch training for MLP. Accuracy, precision, sensitivity and specificity were used as evaluation metrics to assess model performance. The report analyses the strengths and weaknesses of the two models, considering factors such as interpretability and the performance on the selected metrics. It also discusses which model would be better suited for predicting heart disease risk, highlighting the importance of sensitivity in this context.

1 The "Cleveland Heart Disease" dataset

1.1 Attributes of the dataset

The target variable that we want to predict is "target", which represents the diagnosis of heart disease. This is a binary classification problem.

Categorical attributes include sex (1 = male, 0 = female), chest pain type (values 1-4), resting ECG (values 0-2), exercise-induced angina (1 = Yes; 0 = No), st slope (values 1-3), and thalassemia (values 3, 6, and 7).

One-hot encoding is a commonly used method for encoding categorical attributes. This creates a new, binary attribute for each unique category. The value of this attribute is 1 if the corresponding category is present and 0 if it is not.

1.2 Attribute Normalization

Z-score normalization is a suitable method for maintaining variance. It transforms each feature so that it has a mean of zero and a standard deviation of one. Normalization should be done after splitting the data into a training and a test set. This ensures that the statistics from the training set used for normalization do not affect the testing set and lead to a more accurate evaluation.

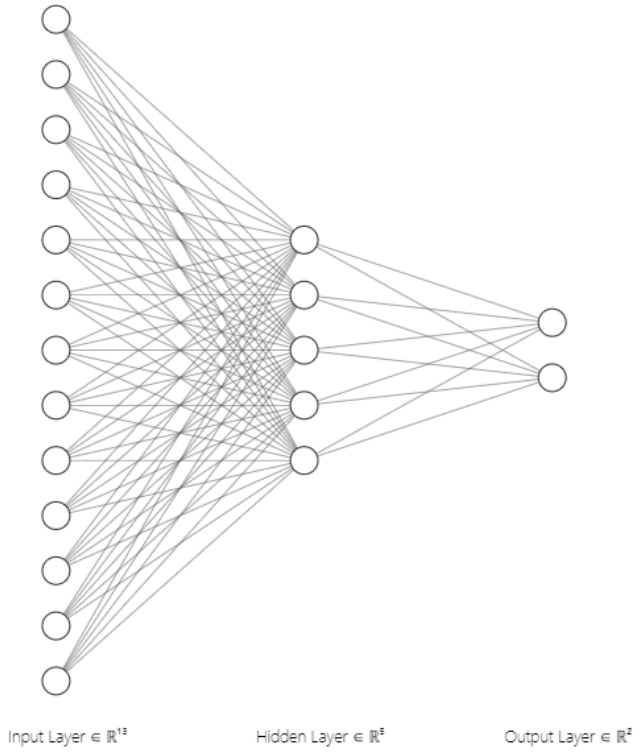
```
1 def normalize(data, mean_vals=None, std_vals=None):
2     if mean_vals is None:
3         mean_vals = np.mean(data, axis=0)
4     if std_vals is None:
5         std_vals = np.std(data, axis=0)
6     normalized_data = (data - mean_vals) / (std_vals + 1e-8)
7     return normalized_data, mean_vals, std_vals
```

2 Predictions using a Multi-layer perceptron

Inputs: (number of samples / number of features) - 13
Outputs: 2 (binary classification)
Mini-batch training involves:

- Shuffling data.
- Dividing data into mini-batches.
- Training on each mini-batch, updating parameters.
- Repeating for all mini-batches in an epoch.

Stop training when training time or epochs limit is reached (early stopping).



2.1 Model evaluation

1. The precision is 85.3%
2. Accuracy is 53.3%
3. Sensitivity is 54.7%
4. Specificity is 45.5%

3 Predictions using a Decision Tree

The data was split into training(0.75) and test (0.25) parts.The same numbers were used in Decision Tree.

- The `calculate_entropy` function calculates the entropy of a set of labels, which is a measure of uncertainty or randomness.

- The `calculate_confusion_matrix` function calculates the confusion matrix, a table that visualizes the performance of an algorithm.
- The `calculate_precision`, `calculate_accuracy`, `calculate_sensitivity`, and `calculate_specificity` functions calculate different metrics of the model's performance using the confusion matrix.
- The `TreeNode` class represents a node in the decision tree. Each node contains a predicted value, the index of the feature used for splitting, the cutoff value for the feature, and the left and right branches of the tree.
- The `DecisionTree` class represents the decision tree itself. It contains methods for preprocessing the data, fitting the model, predicting labels, growing the tree, finding the best split, calculating information gain, splitting the data, traversing the tree, and finding the most common label.
- The `fit` method grows the decision tree using the training data.
- The `predict` method makes predictions on the test data by traversing the tree.
- The `_grow_tree` method recursively grows the tree by finding the best split for the current set of features and labels.
- The `_best_criteria` method finds the best feature and cutoff for splitting the data.
- The `_information_gain` method calculates the information gain of a potential split.
- The `_split` method splits the data based on a feature and cutoff.
- The `_traverse_tree` method traverses the tree to make a prediction for a single data point.
- The `_most_common_label` method finds the most common label in a set of labels.
- Finally, an instance of the `DecisionTree` class is created, the model is fitted to the training data, predictions are made on the test data, and the accuracy, precision, sensitivity, and specificity of the model are calculated and printed.

1. Accuracy is 86.8%

2. Precision is 83.0%

3. Sensitivity is 95.1%

4. Specificity is 77.1%

4 Comparing MLP and DT

Both models were evaluated based on the following metrics:

- **Accuracy**:
  - MLP: 53.3%
  - DT: 86.8%
- **Precision**:
  - MLP: 85.3%
  - DT: 83.0%
- **Sensitivity**:
  - MLP: 54.7%

  - DT: 95.1%

- **Specificity**:
  - MLP: 45.5%
  - DT: 77.1%

4.1 Conclusion

The Decision Tree model is way better than the Multi-Layer Perceptron model in most metrics, especially when it comes to accuracy and sensitivity. Even though the MLP has a suspiciously low accuracy, the better performance of the DT suggests that it's better suited for predicting heart disease risk. To solve the accuracy problem, we need to investigate the MLP implementation and training process further.