Collaboration statement: Course materials, geeks for geeks, medium were used to write some of the contents in the report.

HW3 Report Jinghan Sun 2520665

# Decision Tree Classifier for Heart Disease Prediction

**3.1 Experiment Design**

* **Impurity measure selection**

In this implementation, I chose to use Information Gain based on Entropy as an impurity measure. This measure effectively quantifies randomness or uncertainty in the dataset, providing values between 0 (indicating a pure node) and 1 (indicating maximum impurity for binary classification). Information Gain is calculated as the difference between the parent node's entropy and the weighted average of the child nodes' entropies.

Entropy was chosen among Gini and Classification error because it offers greater sensitivity to probability changes due to its logarithmic nature. For example, with a [0.5, 0.5] probability distribution, entropy gives 1.0 while Gini gives 0.5, making entropy more effective at identifying balanced splits. Additionally, entropy's steeper penalties for impure nodes help create more balanced trees.

* **The way I split each attribute**

I’m making a binary splitting for all the attributes. For numerical attributes, I implement binary splitting with dynamically calculated threshold values. These thresholds are computed as the midpoint between consecutive unique values: (value[i] + value[i+1])/2.

For categorical attributes, I utilize a binary splitting strategy based on equality. This creates splits of the form "attribute = value" versus "attribute ≠ value", evaluating all unique values as potential split points. This approach ensures effective handling of nominal data without imposing artificial ordering.

The split evaluation process incorporates a minimum leaf size constraint of 6 samples to prevent overfitting. The algorithm selects the split that maximizes information gain while maintaining this minimum leaf size requirement. This process continues recursively until reaching either the maximum depth or other stopping criteria.

* **The data structure used**

The Node class serves as the fundamental building block of the tree structure. Each Node instance maintains information about the split feature, threshold value, child nodes, and prediction value when applicable.

Throughout the implementation, I used pandas DataFrame for data manipulation and storage. This choice provides efficient handling of both numerical and categorical data while maintaining separate features (X) and target (y) DataFrames during training and prediction phases. The DataFrame structure offers built-in functionality for data manipulation and type checking, which are useful for processing mixed data types.

* **Additional Techniques**

To prevent overfitting, which I encountered by the beginning of the implementation, I added two hyperparameter: maximum depth (default: 5) and minimum samples per leaf (default: 6). These parameters can be adjusted during model initialization to tune the model's complexity according to specific dataset requirements.

Early stopping conditions are implemented. The algorithm terminates branch growth when reaching maximum depth, when the minimum samples leaf constraint cannot be met, when a node becomes pure (containing samples of a single class), or when no valid splits can be found. These conditions work together to prevent overfitting and ensure computational efficiency.

**3.2 Model Evaluation**

I implemented 10-fold cross-validation as our evaluation strategy. Cross-validation was chosen over simpler methods like hold-out validation because it provides a more robust assessment of model performance by using all available data for both training and validation. The choice of 10 folds represents a standard practice that balances computational efficiency with reliable performance estimation. Another reason I chose 10 folds is I want to know the runtime so when TA doing the same, the runtime is smaller than 15 min.

A screenshot of a computer

Description automatically generated

The overall accuracy of 0.775 (±0.063) demonstrates that the model correctly classifies about 77.5% of cases, with relatively consistent performance across folds. The precision of 0.789 (±0.102) indicates good reliability in positive predictions, while the strong recall/sensitivity of 0.839 (±0.124) shows the model is particularly effective at identifying actual heart disease cases.

The F1-measure of 0.800 (±0.051) confirms a balanced performance between precision and recall, with the lowest standard deviation among all metrics suggesting stable overall performance. However, the specificity of 0.707 (±0.153) indicates some room for improvement in correctly identifying negative cases. The data is imbalanced, so the model tend to predict more result to “yes”. But in the setting of predicting disease, recall > specificity is reasonable because we don’t want to have false negative compared to false positive.

# Cluster Algorithms

1. **K-Means++**

1. Initialize C=empty set; Initialize first centroid randomly from data points

2. Repeat until |C|=k:

- Calculate distance D(x) of each point x to nearest existing centroid, find the minimum

- Select next centroid xk with probability proportional to D(xk,c)²/sum of all D(x,c)^2

- C = the intersect between C and {xk}

Main purpose: Addresses the initialization problem of standard K-means by choosing better initial centroids that are well-spread, produces better results in terms of SSE.

1. **K-Medoids**

1. Randomly select k data points as initial medoids

2. While not converged:

a. Assign each point to nearest medoid

b. For each medoid and non-medoid point o:

- computing the total cost of swapping representative medoid with

- If cost<0, swap medoid with o to form the new set of k representative objects

- Otherwise, don’t swap

Main purpose: More robust to outliers and noise than K-means by using actual data points as cluster centers. Works effectively for small dataset but not scale well for larger ones because computational complexity is high

1. **Bisecting K-Means**

1. Start with all points in single cluster

2. While number of clusters < K:

a. Select a cluster from the list of clusters

b. Apply K-means from i=1 to number of iterations:

- Initialize 2 centroids

- Iterate until add the two clusters from the bisection with the lowest SSE to the list of clusters:

\* Assign points to nearest centroid

\* Update centroids

c. Add resulting two clusters to solution

Main purpose: Creates hierarchical clustering solution, potentially better for non-spherical clusters.