Report HW3

1 Decision Tree Classifier for Heart Disease Prediction

Experiment Design

1. I used Gini impurity change as the impurity measure. The Gini index is computationally less intensive than measures like entropy but is just as robust and performs better than metrics like classification errors. It can capture more subtle changes in the impurity score. Moreover, I did not implement gain ratio because I separated all the attributes into two categories, which prevents the score from being affected by classification groups.

2. I kept consistent in splitting all features into binary groups for categorical attributes. During preprocessing, I converted all categorical variables into numerical values that preserve their original order. Then, the `test_split` function tries each value of the features as a possible split point. The best split point is then selected using the impurity score. For continuous attributes, I also use binary decisions. The `test_split` function tries all the distinct values of the continuous variable to find the best split.

3. I used a customized `DecisionTreeNode` data structure. Each decision and leaf node is an instance of this class. It has variables like `feature`, `threshold`, `left`, and `right`. The `feature` and `threshold` represent the decision boundary of the current node. For decision nodes, `left` and `right` point to the corresponding child nodes. For leaf nodes, `left` and `right` are single values representing class labels (0 or 1).

4. To optimize and speed up the algorithm, I made the `test_split` process more concise by avoiding duplicate values when deciding on the best split. Also, using recursive methods instead of iterative methods makes the code simpler and shorter.

Model Evaluation

I chose cross-validation since the dataset is not big enough. Using cross-validation can provide more accurate evaluation results.

- **Precision**: 0.76 ± 0.06
- **Recall**: 0.75 ± 0.04
- **F1-Score**: 0.75 ± 0.04
- **Specificity**: 0.71 ± 0.08
- **Sensitivity**: 0.73 ± 0.06

The cross-validation results show that the model performs reasonably well, with an average accuracy of 0.73 ± 0.04, meaning it correctly predicts about 73% of the cases. The precision of 0.76 ± 0.06 indicates that when the model predicts a positive case, it is correct 76% of the time. The recall of 0.75 ± 0.04 shows that the model successfully identifies 75% of the actual positive cases. This balance is captured by the F1-score of 0.75 ± 0.04, which combines precision and recall into a single measure. The specificity of 0.71 ± 0.08 means the model is able to correctly identify 71% of negative cases, although the higher variation suggests it is less consistent here. The sensitivity of 0.73 ± 0.06 supports the recall result, showing that the model is good at catching most positive cases. Overall, while the model is balanced and performs well in most areas, the variability in specificity suggests that improvements could help make it more consistent in identifying negative cases.

2 Clustering Algorithms

The standard K-Means algorithm can be heavily influenced by the initial placement of centroids, often resulting in suboptimal clustering outcomes and convergence to local minima. While K-Means++ improves the initialization step, it still inherits the same fundamental disadvantages of K-Means, such as sensitivity to outliers, requiring a predefined number of clusters.

Algorithm 1 K-Means++ Initialization

```
1: **Initialization:** $C = \emptyset$
2: Randomly select a data point $x_0$ to be the first centroid, $C = C \cup \{x_0\}$
3: while $|C| < k$ do
4:     for each point $x_i \notin C$ do
5:         Find the minimum squared distance to any currently selected centroid in $C$:
               $$d(x_i, C)^2 = \min_{x_j \in C} d(x_i, x_j)^2$$
6:     end for
7:     Select a new centroid $x_k$ by randomly choosing a point with probability proportional to $d(x_k, C)^2 / \sum_x d(x, C)^2$
8:     $C = C \cup \{x_k\}$
9: end while
```

K-Medoids is designed to address the sensitivity to outliers that K-Means faces. Instead of using the mean of cluster points as centroids, K-Medoids uses actual data points, making it more robust to noise and outliers. K-Medoids is computationally more intensive than K-Means, especially for larger datasets, as it needs to consider each point as a potential medoid and calculate the distances between all points for every iteration. It may also product empty clusters.

Algorithm 2 K-Medoids Clustering

```
1: Arbitrarily choose $k$ objects in $D$ as the initial representative objects or seeds
2: repeat
3:     Assign each remaining object to the cluster with the nearest representative object
4:     Randomly select a non-representative object $o_{random}$
5:     Compute the total cost $S$ of swapping a representative object $o_j$ with $o_{random}$
           $$S = \sum_{i=1}^{k} \sum_{p \in C_i'} |p - o_i| - \sum_{m=1}^{k} \sum_{p \in C_m} |p - o_m|$$
6:     if $S < 0$ then
7:         Swap $o_j$ with $o_{random}$ to form the new set of $k$ representative objects
8:     end if
9: until no change
```

Bisecting K-Means was developed to improve the clustering quality and provide a hierarchical structure that standard K-Means does not offer. It also solves the problem of empty clusters. It iteratively splits clusters to achieve the desired number of clusters, often leading to more balanced cluster sizes. Although it typically produces better results than standard K-Means, it can still suffer from the same issues with centroid initialization and the assumption of spherical clusters. Additionally, deciding which cluster to bisect and when can be somewhat arbitrary and impact the final clustering.

Algorithm 3 Bisecting K-Means Clustering

```
1: Initialize the list of clusters to contain the cluster with all points
2: while number of clusters $< K$ do
3:     Select the largest cluster to split
4:     for $i = 1$ to number_of_iterations do
5:         Bisect the selected cluster using basic K-Means with $K = 2$
6:     end for
7:     Add the two clusters from the bisection with the lowest SSE to the list of clusters
8: end while
```