Introduction:

This project implements a Random Forest classifier using Java to predict if a patient's health information indicates whether they have heart disease.

Dataset:

- Source: https://www.kaggle.com/datasets/oktayrdeki/heart-disease/data
    - 10,000 patient sample records
- Features: There are 20 features covering a patient's demographics (age, gender), medical information (cholesterol, blood pressure), and lifestyle (smoking, exercise).
- Target: Binary (0 = No heart disease; 1 = heart disease)
- Preprocessing: This was done in a python notebook, *python-analysis/heart_disease_eda.ipynb,* and handles missing values and converts categorical data (gender) into numeric.

Model Implementation

The core logic is distributed across several Java classes in *src/* directory.

DecisionTree.java

This file implements a binary classification tree. It uses the *TreeMatrix*.java helper to compute the entropy-based Information Gain Ratio (IGR) for candidate splits in order to determine the best feature and threshold for splitting nodes. It supports numerical features by using numerical threshold splits and categorical features using set membership splits. There are several stopping criteria that limit the growth of a tree, including hyperparameters like maxDepth and minSamplesSplit, as well as logical criteria such as if there is a low entropy for a node in the tree and thus no significant IGR to be realized from a split. At the end there are leaf nodes that store the majority class label based on samples that reach a given leaf node.

RandomForest.java

This file is an ensemble method that aggregates multiple decision trees together, taking each trees' predicted classification for a given test sample and getting a final predication based on the majority vote across all trees in the forest, as well as the proportion of trees that predicted heart disease in order to get a confidence score.

To improve performance and reduce overfitting, we implement bagging through bootstrap aggregating where each tree is trained on a random subset of the training data. Additionally, at each split only a random subset of features is considered.

HyperparameterTuner.java

This file finds the best hyperparameters that improve a given metric such as accuracy or F1-score. The hyperparameters in this case are number of trees, max depth for each tree, minimum samples to split, and maximum features per split. To find the best hyperparameters, we create a search space that is based on several combinations of hyperparameters. Each combination is tested using k-fold

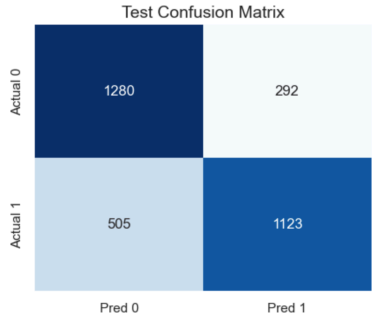
cross-validation and the best hyperparameter set that optimizes for the given metric is used to train the final model.

Results

All model runs are recorded in the */outputs* directory. Initially we optimized for accuracy on the *features_model_ready.csv*. The metrics for this run yielded a high accuracy of 79% but 0% for other important metrics like precision and recall. The more worrisome finding was that our confusion matrix revealed that we failed to get 0 TP. This is when we realized that the model simply predicts no heart disease for everyone. This is because the dataset it was trained on was heavily skewed as out of the 10,000 samples, only 20% of that included patients with disease. From this first run we learned that accuracy can be misleading on an imbalanced dataset, thus shifting our attention to balancing out the dataset.

From our latest run we discovered that balancing the training data through SMOTE oversampling yielded the best results for all metrics we evaluated for. We did this by creating synthetic data that yielded to more samples that were classified as having heart disease. The resulting dataset, *features_model_ready_balanced.csv*, now had 16,000 samples, with 12,800 used for training and 3,200 used for testing. From cross-validation we discovered the best hyperparameters combination to be as follows: (numTrees = 200; maxDepth = 20; minSamplesSplit = 10; maxFeatures = 4).

Below is the evaluation of our final model on the test set:



accuracy	0.750938
precision	0.793640
recall	0.689803
f1	0.738087

We can see that not only did we lose minimal accuracy when compared to the first model which simply guessed the majority class, but now there are significant improvements to the rest of the evaluation metrics as a result of creating a more balanced dataset which allows the model to better learn and detect patterns of the minority class. In particular, we are satisfied with the precision of our model as it shows that when our model predicts heart disease, it is correct almost 80% of the time. The recall is not as high as we would like it to be as this shows that we could only find 69% of call heart disease samples so this is definitely an area we can improve on in the future.