Final Project Report: Stroke Risk Prediction in Rust

A. Project Overview

The goal of this project was to predict the likelihood of a patient having a stroke using their medical data. I wanted to answer the question of whether it's possible to accurately predict stroke likelihood with just basic medical information. To do this, I implemented a decision tree classifier trained using Rust's smartcore crate and created an interactive interface that allows users to input their own data and receive a risk prediction.

The dataset I used came from Kaggle's "Stroke Prediction Dataset," which includes 4981 records of anonymized patients and their health information.

B. Data Processing

I used the csv and serde crates to read in the dataset. Each row was deserialized into a Patient struct, which I created to hold the fields I wanted from the dataset: age, hypertension, heart_disease, ever_married, avg_glucose_level, bmi, and stroke.

One of the biggest challenges in stroke prediction is class imbalance — very few patients in the dataset actually had a stroke. To address this, I oversampled the positive stroke cases by duplicating them three times during training. This significantly improved the recall of the decision tree as now it would be necessary to accurately predict positive strokes in order to maintain a high overall accuracy.

C. Code Structure

The code is organized into three modules:

- data.rs – loads the CSV and defines the Patient struct.
- model.rs – handles feature extraction, oversampling, model training, evaluation, and live prediction. Also the location of my tests.
- main.rs – provides the terminal UI to select evaluation or interactive input mode.

Key Functions and Their Purpose

load_data - Loads patient data from a CSV file

Preprocess_features - Converts patient data into numeric vectors for training

Get_labels - Extracts stroke outcomes as labels

Train_decision_tree - Trains a decision tree on oversampled data

Evaluate_decision_tree - Evaluates model with confusion matrix, accuracy, and recall

Predict_patient - Predicts stroke for a new patient using a trained model

Main Workflow

1. main.rs loads the CSV into memory.
2. The user chooses either:
    - **Evaluation mode**: runs evaluate_decision_tree and prints results.
    - **Interactive mode**: user inputs medical info → struct → features → predict_patient.
3. Both workflows rely on shared logic in model.rs and data.rs.

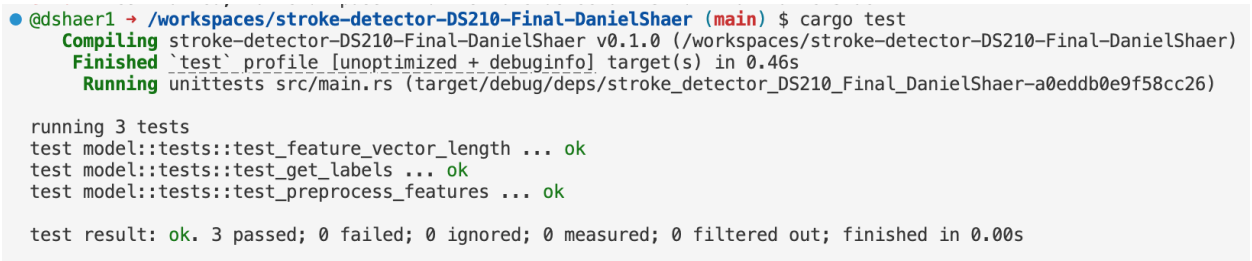
D. Testing

test_preprocess_features – Checks that the feature extraction function returns a vector with exactly 6 elements per patient. This matters because the model expects a consistent input structure, and any change could silently break training or prediction.

test_get_labels – Verifies that the stroke outcome labels are correctly pulled from a list of patients. This is essential for accurate training, as incorrect labels would make the model learn the wrong patterns.

test_feature_vector_length – Confirms that the length of the generated feature vector is exactly 6. This helps catch accidental changes to the input structure, such as adding or removing a feature, which would otherwise go unnoticed.

Image:



E. Results

Input: Cargo run
Output:

```
Loaded 4981 patients

Choose an option:
1. Show evaluation results
2. Enter your own medical info to get stroke risk
```

Now there are two choices, an input of 1 or 2. A different input will terminate the operation.

Input 1: 1
Output: accuracy, recall, and confusion matrix for DTC

```
Decision Tree Results:
Accuracy: 94.85%
Recall (TPR): 97.04%
Confusion Matrix:
TP: 197 | FP: 53
FN: 6 | TN: 889
```

Input 2: 2
Output: sequence of requested entries of medical info and final output of LOW/HIGH stroke risk

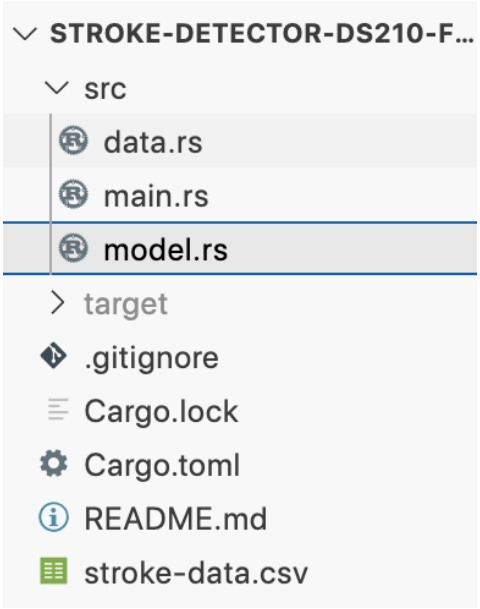
```
2. Enter your own medical info to get stroke risk
2
Enter your age:
75
Hypertension? (0 = No, 1 = Yes):
0
Heart disease? (0 = No, 1 = Yes):
1
Ever married? (0 = No, 1 = Yes):
0
Avg glucose level:
250
BMI:
40

Based on your input, the model predicts: HIGH stroke risk.
```

These results show the model is capable of correctly identifying the majority of stroke cases, even in the presence of class imbalance. The trade-off in slightly more false positives is acceptable given the priority on recall. Additionally, from an ethical standpoint, it's better to have more false positives as opposed to more false negatives, because then you're diagnosing someone with high stroke risk when they really don't as opposed to luring them into a false sense of security with potentially fatal consequences.

F. How to Run

Place stroke-data.csv in the project root, so that the structure looks like this:



Instructions: Build/Run the program in the directory:
/workspaces/stroke-detector-DS210-Final-DanielShaer
and follow any additional instructions from there, making sure your input precisely matches instructed input.
Expected runtime: Near-Instant

G. AI Assistance and Citations

I used chatGPT throughout the project to assist me with syntax and fixing bugs, as well as brainstorming the overall structure for modules and operation. However, all the ideas were my own and whatever was provided by GPT went through many hours of rigorous testing and was based on my own logic.