2. Written Report - Amy Baumel DS210 Museum Decision Tree

A. Project Overview

- Goal: take museum location and type and predict their revenue (based on the subset of data from 2014 that has revenue - about half the data)

- Dataset: The original dataset is 33k rows and 25 columns. https://www.kaggle.com/datasets/imls/museum-directory

B. Data Processing

- Loading: The CSV file is read into the program using standard file I/O operations as well as the csv and serde crates. The functions used to load the data take place in the loading_data module.

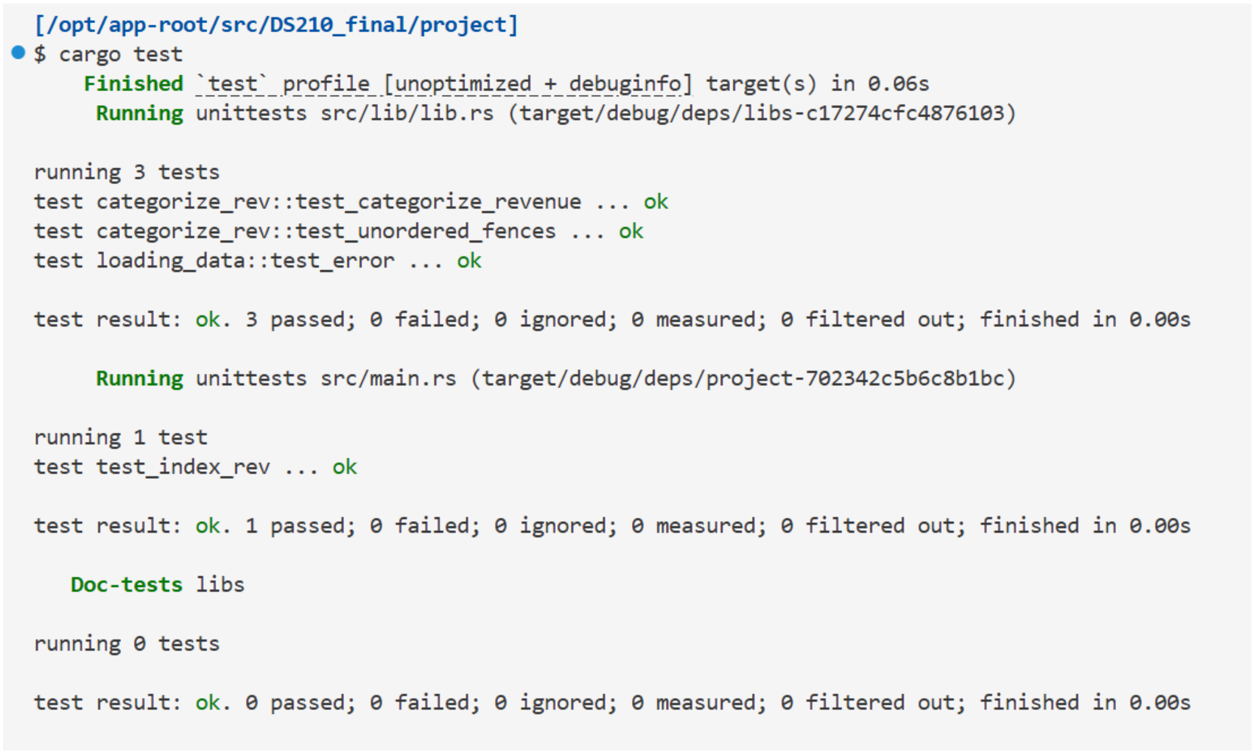
- Cleaning/Transformations:

    - Extracts relevant features from the dataset. Only extract rows that have revenue. (this is done with the loading_data module)
    - Convert categorical and continuous variables into discrete numerical representations suitable for model input. This is done for the museum type with the loading_data module and the categorize module has code that transforms the continuous revenue into discrete ranges with an integer referring to each range.

C. Code Structure

- The categorize module has a function that splits the revenue into different ranges. In the csv file, the museum revenue is a continuous variable, for a decision tree model, I wanted to turn this into a discrete variable. This is how that happens.
- The loading_data module reads a csv file, does a lazy cleaning by only storing complete data (for the features we are interested in) and transforms these rows into a struct with fields being the categorical data and revenue (not categorized yet).
- main.rs: brings everything together, uses both modules to build a decision tree from the transformed data. The transformed data has to be in the correct array form in order to satisfy the dataset form from the linfa crate. The decision tree comes from the linfa_trees::DecisionTree. The program also computes accuracy and demonstrates some examples of using the model to predict revenue. There is a function that matches the prediction to the revenue range that corresponds with it to help user readability of the output.

D. Tests



```
[/opt/app-root/src/DS210_final/project]
$ cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.06s
     Running unittests src/lib/lib.rs (target/debug/deps/libs-c17274cfc4876103)

running 3 tests
test categorize_rev::test_categorize_revenue ... ok
test categorize_rev::test_unordered_fences ... ok
test loading_data::test_error ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

     Running unittests src/main.rs (target/debug/deps/project-702342c5b6c8b1bc)

running 1 test
test test_index_rev ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

   Doc-tests libs

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

test_categorize_revenue in categorize_rev.rs in libs:

This test makes sure revenue is categorized correctly, especially if the revenue is the same as a category bound. If anyone is confused on the rules by which the revenue is categorized (the upper bound of the category is the value at the category # index of the fences), they can check this test to see where values will fall.

test_unordered_fences in categorize_rev.rs in libs:

The categorize function in this file should be able to correctly run even if the fences are out of order. This test checks that. If the fences are out of order and the revenue was categorized like this, values will be put in the wrong categories.

test_error in loading_data.rs in libs:

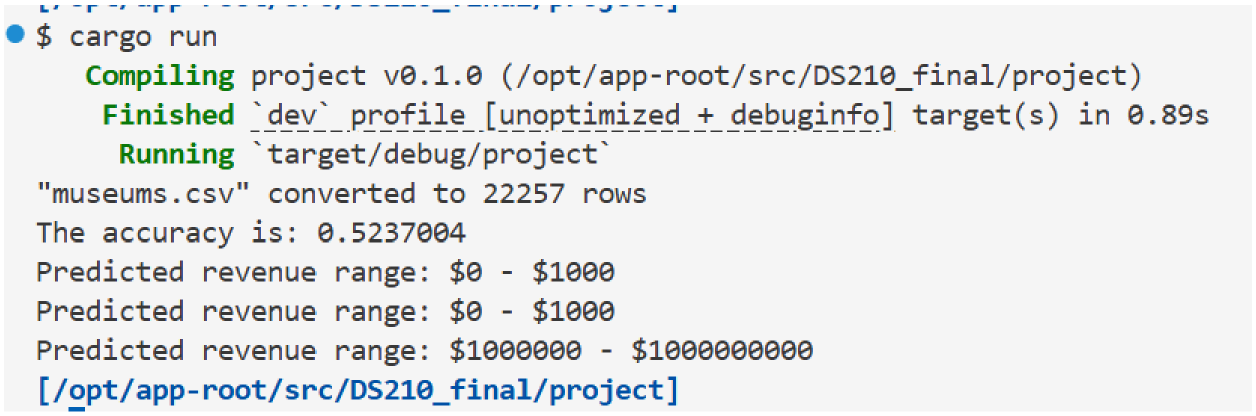
This test checks that the right error will happen when trying to load a file that does not exist. This matters because if it does not do this, the function load() is very wrong.

test_index_rev in main file:

This test is like the test_categorize_revenue test. It checks that the index will be correctly matched back to the correct bounds in fences. This is important because if the format_catetories does not work, the whole program will be misinterpreted into the output that is printed into the terminal.

E. Results

- The output describes the predicted revenue range category for the museum based on the input features and the trained model. For a given (hardcoded) input, the program outputs a string such as: "Predicted revenue range: $100,000 - $1,000,000". The decision tree made to predict revenue achieved an accuracy of 52.37%. The model is not very accurate and should not be depended on for true predictions. The inputs I hardcoded into the program were for Alaska Historical Museum, Barber Vintage Motorsport Museum, and Cooks Natural Science Museum in that order. The predicted revenue ranges are in the screenshot in order.



```
$ cargo run
   Compiling project v0.1.0 (/opt/app-root/src/DS210_final/project)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.89s
     Running `target/debug/project`
"museums.csv" converted to 22257 rows
The accuracy is: 0.5237004
Predicted revenue range: $0 - $1000
Predicted revenue range: $0 - $1000
Predicted revenue range: $1000000 - $1000000000
[/opt/app-root/src/DS210_final/project]
```

F. Usage Instructions

- The program is hardcoded. The places that should be experimented with different values are the fences (try different ranges, different number of ranges, etc) and the example predictions (where the program currently predicts revenue for the Alaska Heritage Museum, Barber Vintage Motorsport Museum, and Cooks Natural Science Museum. The inputs can be found from the csv file, the dataset on the kaggle website, or online (you can even make up a museum if you wanted). The codes for the location types can be found online. (locale code is NCES and the key (and a code lookup map) are on https://nces.ed.gov/programs/edge/docs/NCES_Locale_Framework.pdf) The other codes are found easily like this (FIPS and AAM)

- No command-line arguments or user interaction in the terminal.
- The program executes quickly, under a few seconds.

G. AI-Assistance Disclosure and Other Citations

- AI- Assistance was used for some debugging.
- No collaborators