Abigail Lee

DS 210: “Programming for Data Science”
5.1.25

Final Project Writeup

## I. Project Overview

1. Goal

The goal of this project was to build a decision tree to predict academic status (“Enrolled,” “Graduate,” or “Dropout”) depending on a variety of features. I also wanted to practice assessing the competence of a model, which I did by computing a confusion matrix for my decision tree along with its accuracy and printing the first ten predictions. If the features of an arbitrary student are known, the program is designed to classify the student’s status. The questions I am answering are, “Given arbitrary features, what will the student’s academic status be?” and “How much can we trust our model—that is, how accurate is our tree?”

2. Dataset

The dataset that I used, titled “Predict Students’ Dropout and Academic Success,” was from the University of California Irvine’s Machine Learning Repository. It was designed for classification projects and has 4424 instances and 36 features which are all either floating points or integers. Each instance represents a student and each feature represents a variable about them, while the labels signify their academic status, which are string slices: either “Enrolled,” “Graduate,” or “Dropout.” You can find the link for the dataset here, but it can also be found in the project folder.

## II. Data Processing

1. Loading Data Into Rust

To load my data into Rust, I relied on the File object from Rust’s standard library to open the csv. I then used the ReaderBuilder from the csv crate to iterate through the lines (rows) of the csv. I initially used BufReader but switched to ReaderBuilder so that I could more easily treat the headers differently. I consolidated each row into a single instance of my Student struct, which contained a label (target value) and a vector of features. To populate these fields, I created a vector for each row and indexed the last element of the vector as the label and every preceding element as the features. I then pushed the Student onto a vector and returned a tuple with the feature names and the vector of Students.

2. Cleaning or Transformations Applied

I specified the delimiter of ReaderBuilder to be ";" and then iterated row by row. I extracted the headers using the ReaderBuilder .headers(true) before mapping each header to a String to avoid ownership issues. When dealing with each subsequent row, I first mapped every element to an f64 using a match statement to convert categorical labels and integers. I then created a new features vector by indexing the row to exclude the last element, which was the "Target" header (so not relevant for the features) and set "label" equal to the last element that was excluded.

## III. Code Structure

1. Modules

    a. csv_reading.rs

    The purpose of this module is to handle reading from a CSV file, handling all categorical and integer data, and returning a collection of struct instances that represent my sample. I separated CSV reading into a different file so that I could keep my reading logic distinct from my actual tree or analysis.

        i. Key Functions and Types

        The only function in this module is my read_csv function. The purpose of this function is to read data from a CSV, encode the data numerically, and return a collection of the names of each feature and a collection of instances (individuals that compose my sample). The input to this function was a path to a CSV as a string reference, and the output was a tuple where the first field was a vector of feature names and the second field was a vector of Students (a struct that I will explain later). This function uses ReaderBuilder to iterate over each row of the CSV. I set .has_headers() to true and split along the ";" delimiter. I then mapped the headers to Strings and collected them into a vector and indexed everything except the last element of them to be returned as my feature names. I did this because the last element is the label (dependent variable). I then iterated through each row, and for each row, converted all elements to floats. I set "label" equal to the last element of each row and features equal to every preceding element and collected them into a vector which was a field used for my Student struct. I then pushed each Student (one per row) onto my vector and returned the tuple with both vectors.

    b. tree.rs

This module was designed to build my tree and make predictions on it. I separated this logic from my analysis and reading because this code was most central towards actually constructing the tree and making preliminary predictions.

i. Key Functions and Types

The first type I have is my Student struct, which I made public (along with its fields) so it can be accessed in other modules. Student has a features field which represents the independent variables and is a vector of f64s and a label which represents the dependent variable and is a single f64. Everything is an f64 assuming that the data was input through read_csv.

The first function I have is build_tree, which is designed to construct a decision tree from the data and make a train/test split. It takes in the feature names, a vector of students, and a maximum depth and outputs a decision tree and two DatasetBases from linfa, one for the training split and one for the testing split. The function first pushes all features of all Students onto a vector of flat values and then constructs an array from the flat values where the number of rows is equivalent to the number of Students and the number of columns is the number of features. It also pushes the label of each student onto a vector, which then gets converted to an array. I then constructed a Dataset from linfa using these two new arrays and the feature names before splitting it into 80% train and 20% test. Finally, I initialized the decision tree with a maximum depth equal to the function parameter and fit it on the training data. I then returned a tuple with the tree, the training split, and the testing split.

The second function I have is predict_one_student, which is made to predict the label for a student given its features. The inputs to this function are a decision tree and a vector of features and the output is a label as a String. Labels were previously usizes so that the data types were standardized across features and labels (all were numbers), but for readability I decided to convert back to a String instead using a match statement. This function first converts the features to an array and then makes a prediction using the tree and the new array. It then maps the predicted integer back to a String to be returned. I converted it to a String instead of a string slice to avoid lifetime requirements.

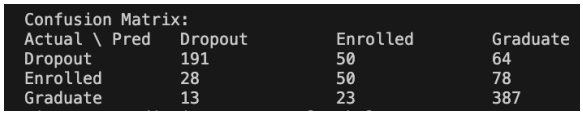
c. metrics.rs

This module was designed to display metrics from the decision tree that was built. This module was separated so that all assessments of accuracy could be separate from reading the CSV and building the tree.

i. Key Functions and Types

The first function I had was print_accuracy, which took in the decision tree, the training split, and the testing split and printed the accuracy evaluated on the training and testing data. It first computed the training accuracy by unwrapping a confusion matrix and then computed the testing accuracy by doing the same, before printing both. I passed both arguments in as references to avoid moving ownership as I would be using the same parameters in later functions.

The second function I had was print_confusion_matrix, which was pretty straightforward; it input references to a decision tree and a test split and printed output. I first printed the labels back from usize to string slices across the top axis of the confusion matrix. I then printed a newline and, for each value, printed the label, printed a row. Each row contained the label for each column, the count of how many times the actual value matched the row label AND how many times it matched the column label. It did this row by row until the confusion matrix was complete.




My next function was print_first_ten_predictions, which took in references to a decision tree and test split and printed the first ten predictions on the test data. I iterated through a range of ten and for each, printed the prediction from the predict method and the actual value from the targets method of the test split.

d. main.rs

In my main.rs, I imported my three modules and the functions I was going to be using from them. I then read from the college_data.csv file I had using read_csv and constructed my decision tree and train test splits using build_tree, using a maximum depth of 6 as an example. However, the tree could be constructed with any variable maximum depth given that it's a parameter of my function. I then used the rand crate to generate a random index to choose an arbitrary student and compared the predicted and actual label. Then, I printed the training and testing accuracy of the tree, the confusion matrix, and the first ten predictions versus the actual labels using my functions.

2. Main Workflow

My program first relies on the csv_reading module to read the CSV into a vector of Students and a vector of features. Then, it uses the function from the tree module to build a decision tree before relying on the metrics module to print all of the metrics of the tree that was built.

*NOTE: in order to count the lines of my code, I repeatedly ran rustfmt to reformat my code.*

## IV. Tests

1. Each Test
    a. test_read_csv_one_row()

        This test made sure that the read_csv() function read the label of a student correctly into an arbitrary struct. I chose a random index and compared it with a value that I knew. This matters to make sure that my reading logic was sound and that the Student structs that were being returned were consistent with the CSV.

    b. test_student_vec_length()

        This test made sure that every single Student was making it into the vector of students that was returned. I checked equality between the number of instances in the CSV and the length of the vector that was returned. This was important to make sure that every individual in the data was being considered in building the tree.

    c. test_predict_one_student()

        This test made sure that the value that was predicted by predict_one_student() was consistent with one of the string references that were actually in the CSV. This was important to make sure that the function was actually referring to labels that existed and was handling the conversion from usize to string reference correctly.

    d. test_print_confusion_matrix_runs()

        This test is simple, as all of the functions in the metrics module don't actually output anything. That being said, I still wanted to write a test for the module as I had two for the csv_reading module and one for the tree.rs module already. This test just makes sure that nothing crashes when print_confusion_matrix() is run.

2. Cargo test output (paste logs or provide screenshots)

```
running 4 tests
test tests::test_read_csv_one_row ... ok
test tests::test_read_csv ... ok
test tests::test_print_confusion_matrix_runs ... ok
test tests::test_predict_one_student ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.58s
```

## V. Results

1. All program outputs (screenshots or pasted)

```
FOR A TREE WITH A DEPTH OF 6:

For student 1973:
The predicted label for this student is: "Graduate"

the actual label for that student is "Graduate"

Train accuracy: 76.05%
Test accuracy: 70.93%

Confusion Matrix:
Actual \ Pred   Dropout         Enrolled        Graduate
Dropout         190             51              64
Enrolled        28              50              78
Graduate        13              23              387

First 10 Predictions vs Actual Labels:
Sample 1: Predicted = 1, Actual = 1
Sample 2: Predicted = 2, Actual = 1
Sample 3: Predicted = 2, Actual = 2
Sample 4: Predicted = 3, Actual = 2
Sample 5: Predicted = 1, Actual = 2
Sample 6: Predicted = 3, Actual = 3
Sample 7: Predicted = 3, Actual = 3
Sample 8: Predicted = 1, Actual = 1
Sample 9: Predicted = 1, Actual = 1
Sample 10: Predicted = 3, Actual = 3
(.venv) MacBook-Air-8:final_project abbylee$
```

2. Interpretation in project context (no need for "groundbreaking" results)

This output demonstrates that for that arbitrary student (1973), the prediction matched the accuracy. It also shared that the accuracy of the decision tree on the training data was 76.05% and the accuracy on the testing data was 70.93%, which may demonstrate slight overfitting on the training data. It also printed the confusion matrix, which demonstrated that the number of accurately classified Students was (190 + 50 + 387 = 627). This makes sense, as the total number of samples was 4424, of which 20% was taken for a test sample, so the total number of test samples was 884. This would make the accuracy equal to the total number of correctly classified individuals over the total tested, which is 627/884 = 70.93%. Finally, It printed the first ten

predictions versus actual labels. This also makes sense, as 7/10 were predicted correctly, which approximates the accuracy of the model.

## VI. Usage Instructions

1. How to build and run your code

My program doesn't take any user input from the command line, but can be compiled using cargo build –release and run using cargo run. Parameters such as the maximum depth of the tree can be edited directly in the program itself and random numbers for generating the individual prediction are generated automatically.

2. Description of any command-line arguments or user interaction in the terminal

N/A

3. Include expected runtime especially if your project takes a long time to run

My project runs in very little time; while I did not specifically time it, for me it printed output in less than a second after taking a little less than 6 seconds to compile.

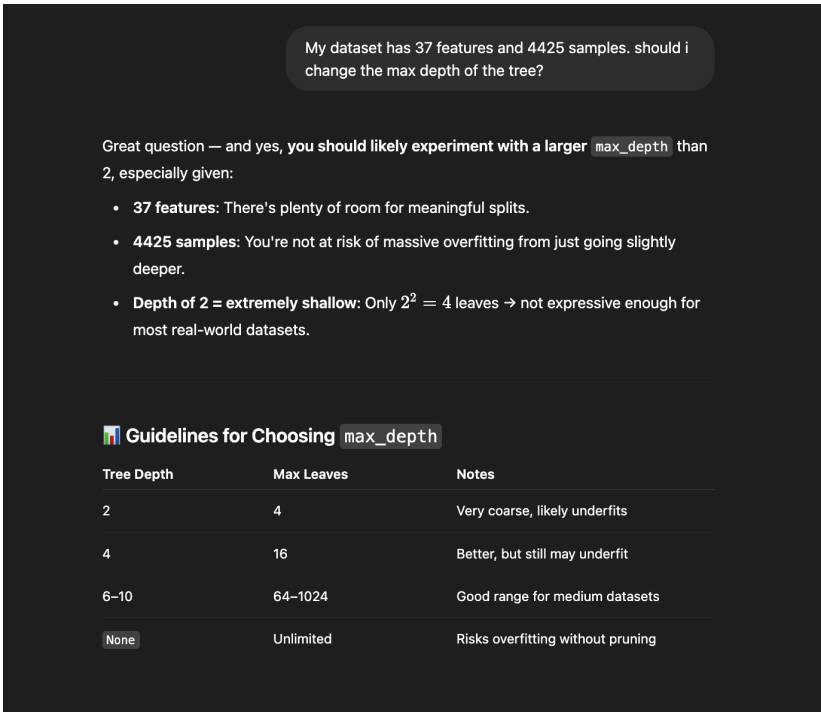
## VII. AI-Assistance and Other Citations

1. Cite any **substantive** chat usage (screenshot or description)

For this project, I relied mostly on lecture notes. However, I did use ChatGPT to debug small issues along the way, and also used it two other times, first to better understand how I should adjust the maximum depth of my tree, and second to interpret the LaTeX code from the lecture.

a. Maximum depth inquiry

Here, I asked ChatGPT about guidelines for deciding the maximum depth of the tree. I followed these guidelines (ChatGPT already understood that my maximum depth at the time was 2) to adjust the depth of my tree. Thinking intuitively, it now makes sense that I would need a greater depth for such a large dataset. Accordingly, I adjusted to 6, which would yield $2^6 = 64$ nodes in total.



b. LaTeX Interpretation

The other time I used ChatGPT substantially was to interpret the code from the lecture that used LaTeX to export to a PNG. I attempted to visualize my tree this way early on, but ended up giving up because I didn't want to install pdflatex.

i'm confused about the latex part of the code; how can i create the png and view it?

Great question — let's walk through the **LaTeX → PDF → PNG** process so you can generate and view the decision tree visualization.

### 🧠 What Your Code is Doing

This line:

```rust
decision_tree.export_to_tikz().with_legend().to_string()
```

...generates **TikZ code**, which is a LaTeX package used to draw diagrams.

Then you're saving it to:

```rust
decision_tree_example.tex
```

So your `.tex` file now contains the full TikZ drawing of the tree.

### ✅ Step-by-Step: From `.tex` to `.png`

#### 1. Compile LaTeX to PDF

In your terminal:

```bash
pdflatex decision_tree_example.tex
```

This should produce:

✅ `decision_tree_example.pdf` in the same directory

> If `pdflatex` is not installed, install it with `brew install mactex` (or use Overleaf — more on that below)

#### 2. Convert PDF to PNG

If you're on macOS, your code already does this via `sips`:

```rust
let sips = Command::new("sips")
    .args([
        "-s", "format", "png",
        "decision_tree_example.pdf",
        "--out", "decision_tree_example.png"
    ])
```

You can also run it manually in the terminal:

```bash
sips -s format png decision_tree_example.pdf --out decision_tree_example.png
```

> 📍 This creates: `decision_tree_example.png`

#### 3. View the PNG

Open the file with:

```bash
open decision_tree_example.png
```

or just double-click it in Finder.