DS 210 Final Project Write Up
Marie Picini

Project Idea and Overview:

My sister and I are debating which Vegas Golden Knights hockey game we want to attend between a few options, and I want to determine which game we will enjoy more. My project aims to find out which we will like more by predicting game-related metrics for the games we are considering, and then calculating an "entertainment score" combining our likes and dislikes to end with a final definitive ranking of which game we should go to. I used decision trees (classification and regression) to make these predictions.

Entertainment Score = 5(Result) + 3(GF) + SF + 1.4(PP%)

Dataset:
I exported my data from the NHL official website and cleaned it extensively: manually, using Google Sheets App Script (javascript), and in Rust when reading in the data. I wanted very specific metrics for this dataset because I wanted the project to be as close to accurate as possible, so, yes I put in a lot of time and effort to cleaning which I won't even get points for because it was in preprocessing before I even used Rust, but I felt it was important to have features which make the most statistical sense as well as the features easiest to track. My dataset (a sample of which is below) has 42 columns and 1702 rows (before adding those for predictions). Each entry represents a regular season game from the 2023-24 NHL season or from the 2024-25 season up until the day I exported the data (12/4/24). Because I wanted to predict future matchup that had not yet occurred, I knew I needed to have pregame metrics, or metrics which I will know at the time of prediction for each future game like season averages of various metrics. Because of the need to have both the in game data for each game and the averages for the full season for each team in each matchup, this created a complex structure for my dataset, hence all the cleaning. I was able to integrate the season averages for each of the metrics I found valuable next to their per game versions as well. This was so I could predict these in game metrics as needed for my entertainment formula, so I neede the real in game metrics as the target and also the averages which I could use to actually predict the values. Further, I had one main categorical variable I wanted to predict as well: Result, which is the game outcome (SOW/W/L/OTL), as this is the most important feature in my formula.
Sample (sorry it isn't in a table, it was too big to fit in one row on one line):

Team Game Date Result(3,2,1,0) P% GF GA GF/GP GA/GP PP% PK% AvPP% AvPK% Net PP% Net PK% AvNet PP% AvNet PK% SF SA Shots/GP SA/GP FOW% AvFOW% Opponent T2P% T2GF T2GA T2GF/GP T2GA/GP T2PP% T2PK% T2AvPP% T2AvPK% T2Net PP% T2Net PK% T2AvNet PP% T2AvNet PK% T2SF T2SA T2Shots/GP T2SA/GP T2FOW% T2AvFOW%

Carolina Hurricanes			12/3/2024		0	0.66	2	4	3.72	2.92	100	100
27.5	82.6	100	100	25.0	84.9	19	28	32.4	25.5	51.8	52.4	Seattle
Kraken	0.481	4	2	2.85	3.0	0	0	14.6	77.4	0	0	13.4
77.4	28	19	28.3	27.6	48.2	49.0						

Code Explanation and Analysis:

I decided to split up my code into the following modules: main.rs (for calling and outputting the predictions and accuracy measures), classification_tree.rs (houses all classification related code), and regression_tree.rs (houses all regression related code). This was the cleanest way I found to handle the differences between what was needed for each type of tree. I researched many different ways online of creating decision trees in Rust, linfa trees, rustlearn, smartcore, and in the end I decided on smartcore as it seemed to be the most extensive and established which would be simple enough to compute what I wanted to compute. The smart core decision trees for regression and classification are quite different in the way they require their inputs to be structured so I had to create two modules which addressed each of their specific needs. However, they are still similar enough that the code in each module mirrors one another, with the same methods and general structuring.

Classification_Tree.rs: This module implements a workflow for processing data (reading, dividing, type correcting) and training a decision tree classifier for the classification of whatever column's index you give it. I chose the Result column for classification, index = 3. Briefly, this module creates and implements the CDataset struct to store features and the target column, onto which I implemented methods to clean, scale, and subset the dataset. The c_scale_features method normalizes numerical columns (ignoring specified categorical cols) by scaling values between 0 and 1, which helped improve model performance. The c_subset method selects specific feature columns for training and predictions, providing the ability to use only some of the features deemed relevant for each tree, but mainly it was used to exclude all in game metrics, which aren't able to be used in training as they wont be present during prediction. Further, the dataset was also converted into the formats needed for the smart core tree classifier (dense matrix or target vector format).

The biggest function, class_read_csv, reads and processes the data from a CSV file (final w: preds.csv). It handles missing values (replacing them with league averages), label-encodes the categorical features via hash map (team names), and converts dates into numeric values l. This function also splits the data into training and prediction datasets, where training data includes features and target labels, while the prediction dataset contains only features for future game predictions, and the prediction set are the first 6 rows of the csv which I manually added to be used for prediction; they are the games Im deciding between as well as a few testers of upcoming games we aren't considering.

Regression_Tree.rs: This module predicts continuous values like Goals For (GF) or Shots For (SF) through regression decision trees. That is the main difference between these modules. Because the regression expects the targets to also be numerical, it takes in f64s not i32s like

the classifier needed. Also, it has the full regression calls and workflow in this module, whereas the actual workflow and calls for the classifier are in the main.rs. The function that houses all that for regression, train_regression_tree, builds a regression model for each target column to predict future values based on the selected features, just as the classifier did. For each prediction, the features are scaled, and the dataset is split into training and testing sets to evaluate the performance (MSE). All the subsequent methods serve the same purpose as their sister functions in the other module.

Main.rs: This file brings all of the trees together and actually calls and outputs what we need (accuracy measures, and predictions). It consists of various components like loading, preprocessing, and training the models and data. The file first imports all necessary libraries and the other modules. Then it has the function classification_workflow which is almost identical to the train_regression_tree function in that it splits and trains the model and outputs and calculates the necessary measures; it overall controls the actual calling and use of the classification tree for result. For the regression trees, it specifies all the inputs for the features used and targets and loops through the indices of the needed targets and creates a tree for each and gives the outputs of the predictions for the prediction dataset and the accuracy (in this case MSE) just like it does for the classification tree in main.

Tests: I have 3 tests, 2 for the classifier methods and 1 for the regression tree. My first test test_c_subset tests that c_subset properly separates test data. My second test test_c_scale_features properly scales test data with numericals to between 0 and 1 and skips categoricals. My third test test_train_regression_tree tests that the tree logic actually works and that predictions are made and the right amount are made on test data.

Prediction and Scoring:
I got the following output for my full project:
Regression tree training successful!
Mean Squared Error of the model: 1.8537
Regression predictions for column index 4 for future games: [4.23, 2.1379310344827585, 2.1379310344827585, 2.1379310344827585, 2.1379310344827585, 2.1379310344827585]
Regression tree training successful!
Mean Squared Error of the model: 784.2398
Regression predictions for column index 8 for future games: [21.57133333333335, 32.69615384615384, 32.69615384615384, 21.57133333333335, 45.70952380952384, 45.70952380952384]
Regression tree training successful!
Mean Squared Error of the model: 44.7206
Regression predictions for column index 16 for future games: [32.5, 32.5, 36.30434782608695, 36.30434782608695, 32.5, 36.30434782608695]
Classification training successful!
Accuracy: 58.38%
Predictions for future games: [2, 0, 3, 1, 0, 0]

I then used this to calculate the entertainment score for each of the 4 games we were between (the last 4 in the predictions listed).

Vs Anaheim: 166.05

Vs New York: 122.68

Vs New Jersey: 184.77

Vs Boston: 190.09

So in closing we should go to the Knights Vs Bruins game!

Sources:

https://smartcorelib.org/user_guide/supervised.html
https://github.com/smartcorelib/smartcore/blob/development/src/ensemble/random_forest_classifier.rs
https://www.geeksforgeeks.org/decision-tree/