DS210 Final Project Write-up

Yifan Zhang

05/04/2025

A. Project Overview

Goal

Analyze which attributes most influence a UFC fighter's performance by ranking feature importances from linear regression.

Dataset

Source: UFC Fighters' Statistics Dataset(ufc-fighters-statistics.csv)

Size: 4111 rows × 18 columns

Link: https://www.kaggle.com/datasets/asaniczka/ufc-fighters-statistics

B. Data Processing

Loading

I used "csv" and "serde" to parse the file, validate headers, skip bad rows(empty and invalid ones), and produce a Vec<FighterRecord>.

Feature Engineering

I first one-hot encode stance and filtered out records with missing or invalid numeric values.

Then I computed:

Ratios: weight/height, reach/height; Rates: takedowns/minute, submissions/minute; Efficiency: submissions per takedown; Win rate; age from date_of_birth; weight-class bucket

Eventually, I normalized all numeric fields to [0,1] to prepare for the regression.

C. Code Structure

1. Modules

main.rs -> Orchestrates: load → preprocess → train → print & plot results

runs the code and gives the output(so that we can use all the modules created and run it to get the output)

io.rs -> CSV I/O + header/row validation → FighterRecord

read the csv file and validates it(so we can easily modify it without affecting other codes)

preprocess.rs -> Cleans data, engineers features, normalizes → CleanRecord

clean and preprocess the data(improving readability and reuse)

model.rs -> Builds feature matrix, fits linear regression, returns sorted (feature, coefficient)

trains the linear regression model on the data(we can change models without affecting other codes) so that it is more concise and short

2. Key Functions & Types (Structs, Enums, Traits, etc)

mod date_format (it contains a method "deserialize") (io.rs)

Purpose: Custom "serde" deserializer to convert a `"YYYY-MM-DD"` string into a "NaiveDate".

Inputs / Outputs: "D: Deserializer<'de, D>" → "Result<NaiveDate, D::Error>"

Logic: Deserialize the incoming field as a "String"; Parse it with "NaiveDate"; Map any parse error into a "serde::de::Error::custom"

FighterRecord (struct) (io.rs)

Purpose: Hold one CSV row in typed fields.

Logic: Uses "serde" + custom date deserializer; skipped rows are logged and dropped.

load_csv(path: &str) -> Result<Vec<FighterRecord>, Box<dyn Error>> (io.rs)

Purpose: Load and validate the CSV file.

Inputs / Outputs: File path → Vec<FighterRecord>.

Logic: Check headers; skip empty or wrong-length rows; deserialize each record via “serde”.

Stance (enum) (preprocess.rs)

Purpose: Represent a fighter’s stance in a type-safe way.

Logic: Represent different stances with different fields.

Method “FromStr”

Purpose: Transfer the strings to the variants

Inputs / Outputs: raw “&str” → “Result<Stance, String>”

Logic: the method “impl FromStr for Stance” matches "Orthodox", "Southpaw", "Switch" to the corresponding variant; Any other string returns “Err("Unknown stance: …")”

WeightClass (enum) (preprocess.rs)

Purpose: Classify a fighter into one weight class based on weight

Logic: the fields correspond to the weight classes annotated with the specific weight ranges

CleanRecord (struct) (preprocess.rs)

Purpose: Store all cleaned and modeled features.

Key fields:

One-hots: is_orthodox, is_southpaw, is_switch; Ratios: weight_height_ratio, reach_height_ratio

Efficiency: submission_per_takedown; Other features like age, significant_strikes_lpm, and so on

preprocess(records: &[FighterRecord]) -> Vec<CleanRecord> (preprocess.rs)

Purpose: Clean raw data and engineer features for modeling.

Inputs / Outputs: slice of FightRecord→ Vec<CleanRecord>.

Logic: Unwrap or drop invalid/NaN fields; One-hot encode stance; Compute ratios, per-minute rates, efficiency, age, win rate, weight class; normalize all numeric features

make_weight_driven_data(path: &str) -> Result<Vec<CleanRecord>, Box<dyn Error>> (pre_process.rs)

Purpose: Helper function to load raw CSV data and immediately run preprocessing.(basically combine the previous functions together)

Inputs / Outputs: File path ("&str") → "Result<Vec<CleanRecord>, Box<dyn Error>>"

Logic: Call "io::load_csv(path)` to get "Vec<FighterRecord>"; Pass the raw records into "preprocess(&raw)"; Return the resulting "Vec<CleanRecord>" or propagate any I/O or parse error)

train_model(records: &[CleanRecord]) -> Result<Vec<(String, f64)>, Box<dyn Error>> (model.rs)

Purpose: Fit a linear regression model and extract feature importances.

Inputs / Outputs: reference of a slice of CleanRecord → "Result<Vec<(String, f64)>, Box<dyn Error >> "

Logic: Build an ndarray matrix of 19 columns (2 stance flags, 7 class flags, 10 numerics); Build target vector from win_rate; Fit linfa_linear::LinearRegression (with intercept); Pair coefficients with feature names; Sort by descending absolute value

plot_importances(results: &[(String, f64)]) -> Result<(), Box<dyn std::error::Error>> (main.rs)

Purpose: Plot and save a horizontal bar chart showing each feature's coefficient.

Inputs / Outputs: a slice of `(feature_name, coefficient)` pairs → Result<(), Box<dyn std::error::Error>>

saves “feature_importances.png” and returns `Ok(())` (doesn’t return anything)

Logic: Split “results” into “names” (Y labels) and “coefs” (coefficients); Compute X-axis range with 10% padding around the extreme coefficients; Prepare the drawing area; Build a Cartesian chart: numeric X vs. integer Y (0..count); Configure mesh: disable grid, place exactly “count” Y-ticks, format each tick as its feature name; Draw one filled rectangle per feature, spanning from 0 to coefficient on its Y-row; Return “Ok(())”

main() -> Result<(), Box<dyn Error>>   (main.rs)

Purpose: load data, preprocess, model training, and visualize(the main function we execute)
Inputs / Outputs: noting → nothing(console printout of feature importances and “feature_importances.png” file)

Logic: Parse CLI argument for CSV path; Call “load_csv(path)` → `Vec<FighterRecord>”; Call “preprocess(&raw) → Vec<CleanRecord>”; Call “train_model(&cleaned) → sorted Vec<(String, f64)>”; Print each “(feature, coefficient)” to stdout; Call “plot_importances(&results)” to save a bar-chart PNG

3. Main Workflow

IO (CSV) → FighterRecord → preprocess → CleanRecord → model → coefficients → plot

In detail: (it all happens in main.rs)

reads file path → calls “load_csv” in “io.rs” → calls “preprocess” in “preprocess.rs” → calls “train_model” in “model.rs”→ prints coefficients → calls “plot_importances” in “main.rs”, which generates a labeled horizontal bar chart “feature_importances.png”.

D. Tests

Cargo test output:

```
    Finished `test` profile [unoptimized + debuginfo] target(s) in 1.78s
     Running unittests src/main.rs (target/debug/deps/final_project-97f2b27e10bd9d60)

running 3 tests
test tests::test_train_model_simple ... ok
test tests::test_load_csv ... ok
test tests::test_preprocess_filters_and_features ... ok

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

Test explanations:

test_load_csv: verifies CSV header matching, parsing of numeric, string, and date fields.

It matters because we want to make sure it reads the features how we want it to.

test_preprocess_filters_and_features: ensures filtering of invalid rows and that normalized numeric fields collapse to 0.0 when only one record remains; stance one-hots correct.

It matters because we don't want any empty row/column to influence our regression process.

test_train_model_simple: I gave it tiny two-record datasets where weight_height_ratio perfectly predicts win_rate, checks coefficient > 0.

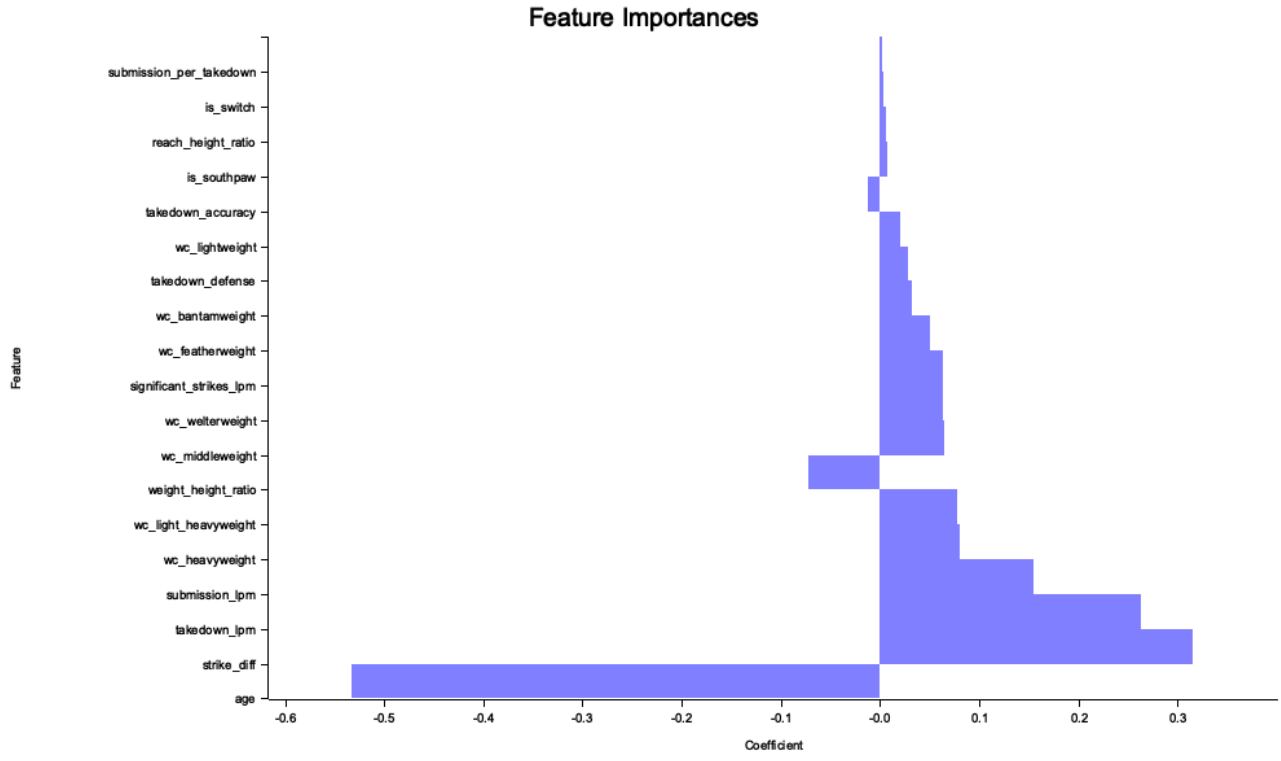
It matters because we need to make sure that the program recognizes the positive and negative influences.

E. Results

All program outputs:

```
Feature importances:
age                          -0.5323
strike_diff                   0.3148
takedown_lpm                  0.2626
submission_lpm                0.1543
wc_heavyweight                0.0806
wc_light_heavyweight          0.0775
weight_height_ratio          -0.0723
wc_middleweight               0.0647
wc_welterweight               0.0636
significant_strikes_lpm       0.0634
wc_featherweight              0.0505
wc_bantamweight               0.0321
takedown_defense              0.0282
wc_lightweight                0.0202
takedown_accuracy            -0.0117
is_southpaw                   0.0070
reach_height_ratio            0.0062
is_switch                     0.0030
submission_per_takedown       0.0015
Wrote feature_importances.png
```

Visualization:



Interpretation:

My hypothesis was that the reach relative to height would contribute the most to win_rate since that allows a fighter to attack the opponent more. However, it is not the case. Conversely, reach relative to height barely does anything to improving the fighter's performance.

Here are some interesting interpretations:

Older age has the strongest negative correlation with the winning rate. Getting old is a fighter's biggest opponent. It makes sense because a fighter's body performance would naturally decline as he gets old.

As strike difference(significant strikes landed per minute minus significant strikes absorbed per minute) is the feature that positively contributes the most, we can say that improving the striking accuracy and having a better defense is the key to better performance. It makes sense because mixed martial arts is all about hitting the opponent while not getting hit.

Using orthodox as the baseline, we can see that fighting southpaw and switching stances constantly both have minimal influence on the winning rate, although fighting southpaw might have a small advantage. It is a little surprising because I thought that the stances are important since they determine which hand the fighter has in front. But it also makes sense because the other features, such as the significant strike differences and submissions landed, are more direct influences on the outcome of the fights.

Of all the weight classes, lightweight contributes the least positively to the winning rate. It means that lightweight is the hardest weight class in UFC to compete in. It makes perfect sense because the lightweight division is generally considered the most competitive and entertaining weight class with the most popular fighters such as Charles Oliveira, Dustin Poirier, Justin Gaethje, Islam Makhachev, and Khabib Nurmagomedov.

Of all the features, the one with the least absolute coefficient is submission per takedown, which is the submission attempt rate. It is a little surprising because there are some fighters such as Khabib Nurmagomedov who are known for going for submissions after takedowns and have high winning rates. But it makes sense because only a few fighters in UFC are good at wrestling and submissions. The majority of fighters are still strikers and don't try takedowns or submissions. Data regarding striking is more representative. The reason for this mismatch can also be that we only have data regarding submission attempts instead of successes. The submission success rate might say more to the winning rate.

F. Usage Instructions

1. **Build**

   First download the original csv file and then put it in the same folder as the directory we are working on.

   Then write the “io.rs” module to read the file.

   Then write the “preprocess.rs” file to process the file and extract features.

   Then write the “model.rs” module to train the models.

   Finally, write the “main.rs” to create the visualization function and the main function to connect all the functions together to get the output. Then write the test cases to make sure the individual functions all work.

2. **Run**

   Copy and paste the file paths to the path section in “main.rs”, and then type “cargo run –release” in the terminal.

   Outputs

   Console: list of features + coefficients.

   PNG: feature_importances.png in the file.

3. **Expected runtime**

   While using “cargo run –release”, the expected runtime is below one second, usually 0.23 seconds.

G. AI-Assistance Disclosure and Other Citations

https://docs.rs/chrono/latest/chrono/naive/struct.NaiveDate.html

I used this website for reference to date representations.