Project: Titanic Survival

Git Repository:
https://github.com/harrrilan/DS210_official_repo/tree/main/Project_TitanicSurvival

Description:

- Goal Predict passenger survival (Survived ∈ {0, 1})
- Source Kaggle “Titanic Dataset”:https://www.kaggle.com/datasets/yasserh/titanic-dataset
- Records 891 train, 418 test, 12 raw features
- As one of the classic datasets in data science, this dataset includes both continuous and discrete variables along with clearly defined outcome variables.

Type	Feature(s)	Notes
ID	PassengerId	Unique key
Target	Survived	0 = No, 1 = Yes
Categorical	Pclass, Sex, Ticket, Cabin, Embarked	Ordinal ↔ nominal mix
Ordinal Counts	SibSp, Parch	# relatives aboard
Text	Name	Used for title extraction
Numeric	Age, Fare	Continuous

File Structure:

```
/titanic
├── Cargo.toml
├── Cargo.lock
├── /data
│   ├── processed_titanic.csv
│   └── Titanic-Dataset.csv
└── /src
    ├── main.rs         # entry point, defines modules
    ├── evaluation.rs   # metrics and evaluation functions
    ├── regression.rs   # logistic regression model
    ├── scaling.rs      # feature scaling functions
    ├── split.rs        # train/test split functions
    └── load_file.rs    # CSV loading
```

Pre-Processing on Python

Python: Basic Processing

- One Hot encoding was done on python

```
df['Embarked'] = df['Embarked'].map({'C': 0, 'Q': 1, 'S': 2})

df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
```

- Any rows that contains NaN was removed
- Features that are irrelevant were removed
    - 'PassengerId'
    - 'Name'
    - 'Ticket'
    - 'Cabin'

load_file.rs

Module Purpose:

This module provides basic CSV file loading and inspection tools for numerical data, using ndarray for structured matrix representation. It's organized for reusability in numeric computing or ML preprocessing tasks.

Main Workflow Overview

```
Shape: (712, 8)
Showing first 5 of 712 rows, 8 columns:
[0.0, 3.0, 0.0, 22.0, 1.0, 0.0, 7.25, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[1.0, 1.0, 1.0, 38.0, 1.0, 0.0, 71.2833, 0.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[1.0, 3.0, 1.0, 26.0, 0.0, 0.0, 7.925, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[1.0, 1.0, 1.0, 35.0, 1.0, 0.0, 53.1, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[0.0, 3.0, 0.0, 35.0, 0.0, 0.0, 8.05, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
```

1. After data clean from python, above is the shape, and the head of the dataframe.
2. 712 rows, and 8 columns.
3. A CSV file is read using load_csv, which parses and validates it into a consistent 2D Array2<f32>.
4. The resulting array can then be inspected using print_head.
5. Tests ensure load_csv behaves correctly across edge cases like inconsistent row lengths, custom delimiters, and optional headers.

Function: load_csv

- **Purpose:** Load a CSV file with numerical values into an ndarray::Array2<f32> matrix, supporting optional headers and configurable delimiters.

- **Inputs:**
  - path: &str – path to the CSV file
  - delimiter: char – character used to separate values (e.g., ',', ';')
  - has_headers: bool, whether to skip the first row as headers
- **Outputs**
  - Result<Array2<f32>, Box<dyn Error>> – a 2D numerical array or an error.
- **Core Logic**
  - Open the file and wrap it in a buffered reader.
  - Iterate through lines, skipping headers if necessary.
  - For each row:
    - Parse and trim each column as a f32.
    - Check that all rows have the same number of columns.
  - Accumulate all data into a flat Vec<f32>.
  - Construct a 2D Array2 from shape and data.
- **Key Components**
  - ndarray::Array2 – for representing numerical matrices
  - Error handling via Result and early returns with "?"

Function: print_head

- **Purpose:** Print the first n rows of an Array2<f32> for quick inspection.
- **Inputs**
  - array: &Array2<f32> – reference to the numeric matrix
  - n: usize – number of rows to display
- **Outputs**
  - None (side-effect: prints to stdout)
- **Core Logic**
  - Determine min(n, total_rows)
  - Use .rows() iterator to access and print the top rows

Test Module:

- **Purpose of Tests:** To validate that load_csv and print_head function correctly under various conditions, including:
  - Correct parsing with/without headers
  - Handling different delimiters
  - Erroring on malformed input
  - Safe execution of print logic

Key Test Points

- **load_csv_with_headers:** Verifies correct parsing with headers skipped.
- **load_csv_without_headers_semicolon_delim:** Checks support for custom delimiter without headers.

- **load_csv_inconsistent_columns_should_error:** Ensures a mismatched row length triggers an error.
- **print_head_does_not_panic:** Confirms print_head executes without panicking or crashing.

split.rs

Module Purpose

This module provides a utility function to split a 2D numerical dataset (in ndarray::Array2<f32>) into training and testing subsets, with optional reproducibility via a random seed. It's designed for use in machine learning or statistical modeling workflows.

Main Workflow Overview

1. Input data (a 2D array) is split into training and test sets using randomized row selection.
2. Optional seed allows reproducible splitting (important for experiments).
3. The select function from ndarray extracts the appropriate rows.
4. Tests validate correctness, reproducibility, uniqueness, and edge cases.
5. Results:
    - Train shape: (570, 8)
    - Test shape: (142, 8)

Function: train_test_split

- **Purpose:** Randomly split a 2D dataset into training and testing subsets, preserving row integrity.
- **Inputs**
    - data: &Array2<f32> – dataset where each row is a data sample
    - test_ratio: f32 – fraction of the dataset to use as the test set
    - seed: Option<u64> – optional seed to make the shuffle deterministic
- **Outputs:**
    - (Array2<f32>, Array2<f32>) – a tuple containing the train set and test set

- **Core Logic & Key Components**
    - Assert that test_ratio is in (0.0, 1.0)
    - Compute the number of test samples based on the total number of rows
    - Shuffle the row indices using a seeded RNG (StdRng) for reproducibility
    - Use ndarray::select with the shuffled indices to split data:
        - First n_test rows → test set
        - Remaining rows → train set
- **Relevant Libraries**
    - ndarray::Array2 – represents 2D numerical arrays

- rand::{SeedableRng, SliceRandom} – used for shuffling with optional determinism

Test Module

- **Purpose of Tests:** To ensure train_test_split correctly:
  - Respects size proportions
  - Supports deterministic and stochastic behavior
  - Produces disjoint splits with complete coverage
  - Works even with minimal data

Key Test Points

- **split_sizes_are_correct:** Validates that the number of training and test rows matches expectations from test_ratio.
- **reproducible_with_same_seed:** Ensures deterministic behavior when the same seed is used.
- **different_seeds_give_different_split:** Confirms that different seeds result in different splits (statistically very likely).
- **no_overlap_and_no_missing_rows:**
  - Checks that:
    - Training and test sets have no overlapping rows
    - Together, they represent the full dataset (no rows lost or duplicated)
- **handles_tiny_inputs:** Tests edge behavior with very small datasets (e.g., one row).
- **Helper: row_set:**
  - Converts Array2<f32> into a HashSet<Vec<NotNan<f32>>> for set comparisons to detect overlaps or missing data.
  - Uses ordered_float::NotNan to allow f32 inside a HashSet.

scaling.rs

Module Purpose

This module provides a standardization function (standard_scale) to normalize feature columns of training and test datasets. It centers each feature to have zero mean and unit variance using training statistics, while leaving the label column untouched.

Main Workflow Overview

1. Input training and test datasets are passed by mutable reference.
2. Columns are standardized column-wise using training data stats.
3. Both datasets are updated in-place.
4. The function returns the means and stds used (excluding label column).

Function: standard_scale

- **Purpose:** Normalize feature columns (excluding column 0) in both training and test datasets using the training set's mean and standard deviation.

- **Inputs**
  - train: &mut Array2<f32> – Training data (with labels in column 0)
  - test: &mut Array2<f32> – Test data (same format, must match column count)
- **Outputs:** Vec<(f32, f32)> – Vector of (mean, std) per feature column (excluding labels)
- **Core Logic**
  - Assert train and test have equal column counts.
  - For each column **starting from index 1**:
    - Compute mean and standard deviation on training data.
    - Replace std with 1.0 if it is zero (to prevent division-by-zero).
    - Subtract mean and divide by std on both train and test columns.
    - Collect (mean, std) into a stats vector.
  - Return the stats for potential reuse or verification.
- **Key Components**
  - train.column_mut(col) and test.column_mut(col) for in-place mutation
  - .mean() and .std(0.0) from ndarray
  - First column (label) is skipped, making it safe for supervised learning tasks

Test Module

Purpose of Tests

- Feature scaling behaves correctly across normal and edge cases.
- The label column remains unaltered.
- Column mismatch causes a panic as expected.

Key Test Points

- **standard_scale_basic**
  - Validates:
    - Only feature columns (index 1+) are scaled.
    - After scaling, training features have ≈0 mean and ≈1 std.
    - Label column is untouched.
- **standard_scale_constant_feature**
  - Ensures:
    - Constant feature column (zero std) is safely handled.
    - All values in such columns become 0.0 post-scaling.
    - Reported std is overridden to 1.0.

- **standard_scale_panics_on_column_mismatch:** Tests panic when train/test datasets have mismatched column counts.
- **Helper: close(a, b, tol):** Utility for checking approximate float equality (used in assertions).

regression.rs

Module Purpose

This module provides functionality for fitting and evaluating a logistic regression model using the linfa machine learning framework. It includes dataset preparation (label-feature splitting), model training, and performance evaluation. It assumes the label is stored in column 0 and features start from column 1.

Main Workflow Overview

- The input dataset (as Array2<f32>) is split into features and labels.
- A logistic regression model is trained using the linfa-logistic crate.
- The trained model can then be used to make predictions.
- These predictions are evaluated using an external evaluate function to return standard classification metrics.
- Output:
    - Fitting logistic regression model…
    - Model training complete.

Function: fit_logistic_model

Purpose: Train a logistic regression model using the provided dataset.

Inputs

- data: &Array2<f32> – A dataset where column 0 is the label and remaining columns are features
- max_iter: u32 – Maximum number of iterations for model convergence

Outputs: Result<Model, Box<dyn Error>> – A fitted logistic regression model or error

Core Logic

- Calls split_features_labels() to prepare features and labels
- Constructs a Dataset from features and labels
- Fits a logistic model with a configurable iteration limit

Key Components

- linfa::Dataset – Provides a standard format for ML data
- LogisticRegression::fit() – Fits the model
- Error handling wraps model fitting failures

Function: split_features_labels

Purpose: Split a full dataset into features and integer labels.

Inputs: data: &Array2<f32> – Dataset with shape (rows, ≥2 columns)

Outputs

- Result<(Array2<f32>, Array1<i32>), Box<dyn Error>> – Tuple of (features, labels)

Core Logic

- Validates at least two columns (label + one feature)
- Slices out column 0 as labels, and columns 1.. as features
- Converts label values to i32

Key Components

- slice_axis(Axis(1), Slice::from(..1)) – Extract label column
- into_shape(nrows) – Flattens label matrix to a vector
- .mapv(|x| x as i32) – Type conversion for compatibility with Linfa

Function: evaluate_model

Purpose: Evaluate a fitted model on a test dataset and return performance metrics.

Inputs

- model: &Model – A trained logistic regression model
- data: &Array2<f32> – Dataset for evaluation (same format as training)

Outputs: Result<Metrics, Box<dyn Error>> – Evaluation metrics struct

Core Logic

- Splits test data into features and labels
- Generates predictions from the model
- Evaluates predictions using evaluate(&predictions, &labels)

Key Components

- model.predict(&features) – Generates predicted labels
- evaluate() – Computes confusion matrix and related metrics

Test Module

Purpose of Tests

- To validate correctness of dataset splitting, model training, and end-to-end behavior, including edge cases like invalid inputs.

Key Test Points

- **split_features_labels_basic**
  Verifies:
  - Features and labels are extracted correctly
  - Feature shape and label content match expectations
- **split_features_labels_single_column_errors:** An error is returned when there is no feature column (only labels)
- **fit_and_evaluate_on_simple_separable_data**
  Tests full pipeline:
  - Fits a model on clearly separable binary data
  - Predicts on the same dataset
  - Asserts confusion matrix shows perfect classification
  - Verifies evaluate() works correctly

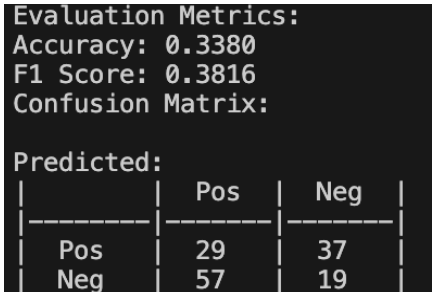
evaluation.rs

Module Purpose

This module implements evaluation logic for binary classification tasks. It provides utilities to compute classification metrics including accuracy, F1 score, and confusion matrix components from predicted and true labels.

Main Workflow Overview

1. Given predictions and ground-truth labels (Array1<i32>), the module computes counts of TP, TN, FP, and FN.
2. These are used to derive standard classification metrics: accuracy and F1 score.
3. Results are returned in a Metrics struct that includes a ConfusionMatrix.
4. Outputs:



Struct: ConfusionMatrix

Purpose: Represents the 2×2 breakdown of predicted vs. actual outcomes.

Fields:

- tp: usize – True positives
- tn: usize – True negatives
- fp: usize – False positives
- fn_: usize – False negatives

Key Features:

- Implements Display trait to print a formatted confusion matrix table.

Struct: Metrics

Purpose: Bundles commonly used evaluation metrics.

Fields:

- accuracy: f32 – Overall classification accuracy
- f1: f32 – F1 score (harmonic mean of precision and recall)
- confusion: ConfusionMatrix – Underlying raw classification counts

Function: evaluate

Purpose: Evaluate predicted binary labels against ground-truth labels to derive classification metrics.

Inputs

- pred: &Array1<i32> – Predicted labels (0 or 1)
- truth: &Array1<i32> – True labels (0 or 1)

Outputs: Metrics – Struct containing accuracy, F1 score, and confusion matrix

Core Logic

- Validates equal length between predictions and truth arrays
- Calls internal helper confusion_counts()
- Computes:
    - accuracy = (tp + tn) / total
    - precision = tp / (tp + fp)
    - recall = tp / (tp + fn_)
    - f1 = 2 * precision * recall / (precision + recall)
- Handles division-by-zero gracefully by returning 0.0

Key Components

- Uses ndarray::Array1 for vectorized data
- Internal numeric checks to prevent division errors
- Returns results as a rich struct for downstream consumption

Function: confusion_counts (internal helper)

Purpose: Compute raw counts for TP, TN, FP, FN by comparing predictions to ground truth.

Inputs

- pred: &Array1<i32>
- truth: &Array1<i32>

Outputs: (usize, usize, usize, usize) – Tuple of (tp, tn, fp, fn_)

Test Module

Purpose of Tests

- Validate correctness of confusion matrix logic and metric computation, including edge cases like perfect prediction.

Key Test Points

- **`confusion_counts_matches_expected`**
  Confirms that the count of TP, TN, FP, and FN matches expected values for a known prediction/truth pairs
- **`evaluate_on_perfect_predictions`**
  Ensures that:
    - Accuracy and F1 are both 1.0 when predictions are perfect
    - No false positives or false negatives are reported

main.rs

Module Purpose

This is the entry point of the application. It orchestrates the complete machine learning pipeline, including loading the dataset, preprocessing (standardization), training a logistic regression model, and evaluating its performance. It uses modular components to separate concerns like data loading, splitting, scaling, model fitting, and evaluation.

Main Workflow Overview

1. Print the current working directory (for debugging/traceability).
2. Load a CSV file (data/processed_titanic.csv) into an ndarray::Array2<f32>.
3. Print the shape and preview of the data using print_head.
4. Split the data into training and test sets using a fixed random seed.
5. Apply standard scaling (zero mean, unit variance) to feature columns.
6. Fit a logistic regression model on the training set.
7. Evaluate the model on the test set and print metrics:
    - Accuracy
    - F1 score
    - Confusion matrix (TP, TN, FP, FN)

Function: main

Purpose: Execute the entire ML pipeline using functions defined in modular subcomponents.

Inputs

- None (uses hardcoded relative path to input CSV)

Outputs

- Result<(), Box<dyn Error>> – Returns Ok(()) on success or propagates errors on failure

Core Logic

- Calls load_csv to load the dataset
- Uses train_test_split with a 0.20 test ratio
- Calls standard_scale to normalize features
- Trains model using fit_logistic_model
- Calls evaluate_model to get performance metrics
- Prints metrics to standard output

Key Components

- load_csv, print_head from load_file module
- train_test_split from split module
- standard_scale from scaling module
- fit_logistic_model, evaluate_model from regression module
- evaluate (used internally by evaluate_model) from evaluation module

Test Module

Purpose of Tests

Provide an end-to-end "smoke test" that ensures the entire main pipeline runs successfully using a small, synthetic dataset. It ensures

- File system assumptions are met (data/processed_titanic.csv exists)
- No runtime panics or logic errors occur in the pipeline
- Integration between modules remains valid

Key Test Points

- **make_dummy_project_tree:** Creates a temporary project structure with a minimal Titanic-like CSV in the correct file path (data/processed_titanic.csv). Used to simulate a real-world file-based workflow.

- **main_smoke_test**
    - Temporarily switches to the dummy project directory.
    - Calls the main() function exactly as in production.
    - Asserts that main() returns Ok(()), meaning the pipeline completes end-to-end without errors
    - Restores original working directory after test.

Test Outputs

```
running 18 tests
test evaluation::tests::confusion_counts_matches_expected ... ok
test evaluation::tests::evaluate_on_perfect_predictions ... ok
test regression::tests::split_features_labels_basic ... ok
test load_file::tests::print_head_does_not_panic ... ok
test regression::tests::split_features_labels_single_column_errors ... ok
test scaling::tests::standard_scale_constant_feature ... ok
test scaling::tests::standard_scale_basic ... ok
test scaling::tests::standard_scale_panics_on_column_mismatch - should panic ... ok
test split::tests::handles_tiny_inputs ... ok
test split::tests::different_seeds_give_different_split ... ok
test split::tests::no_overlap_and_no_missing_rows ... ok
test split::tests::split_sizes_are_correct ... ok
test split::tests::reproducible_with_same_seed ... ok
test regression::tests::fit_and_evaluate_on_simple_separable_data ... ok
test load_file::tests::load_csv_without_headers_semicolon_delim ... ok
test load_file::tests::load_csv_inconsistent_columns_should_error ... ok
test load_file::tests::load_csv_with_headers ... ok
test tests::main_smoke_test ... ok

test result: ok. 18 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s
```

Results:

```
Current dir: "/Users/lanhakuken/Documents/GitHub/DS210_official_repo/Project_TitanicSurvival"
Shape: (712, 8)
Showing first 5 of 712 rows, 8 columns:
[0.0, 3.0, 0.0, 22.0, 1.0, 0.0, 7.25, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[1.0, 1.0, 1.0, 38.0, 1.0, 0.0, 71.2833, 0.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[1.0, 3.0, 1.0, 26.0, 0.0, 0.0, 7.925, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[1.0, 1.0, 1.0, 35.0, 1.0, 0.0, 53.1, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
[0.0, 3.0, 0.0, 35.0, 0.0, 0.0, 8.05, 2.0], shape=[8], strides=[1], layout=CFcf (0xf), const ndim=1
Train shape: (570, 8)
Test  shape: (142, 8)
Fitting logistic regression model...
Model training complete.
Evaluation Metrics:
Accuracy: 0.3380
F1 Score: 0.3816
Confusion Matrix:

Predicted:
|        |  Pos  |  Neg  |
|--------|-------|-------|
|  Pos   |  29   |  37   |
|  Neg   |  57   |  19   |
```

- Struggled in trouble shooting why it was different from the result from Sckit Learn that I got under the same condition.
- Using the whole PCAed data frame might performed better.

Usage Instructions:

- Line 23:

```
let (mut train, mut test) = train_test_split(&data, 0.20, Some(42));
```

  - We can change the proportion of train and test set.
- Line 31:

```
let model = fit_logistic_model(&train, 500)?;
```

  - We can change the maximum number of iterations
- Cargo run

AI Usage:

✅ Step 1: Understand the Data Format

From your `train_test_split` implementation in `split.rs`:

rust Copy Edit

```rust
pub fn train_test_split(
    data: &Array2<f32>,
    test_ratio: f32,
    seed: Option<u64>,
) -> (Array2<f32>, Array2<f32>)
```

You return two raw `Array2<f32>` matrices (`train`, `test`). These contain both features and the target column, with **no separation of X and y** yet.

🔍 Requirement from Linfa

Linfa's `LogisticRegression` requires a `DatasetBase`:

rust Copy Edit

```rust
pub type DatasetBase<X, Y> = DatasetBase<ArrayBase<S, D>, ArrayBase<S2, D2>>;
```

So you'll need to:

- Extract the last column as labels (`y`)
- Use the remaining columns as features (`X`)

✅ Final Interface for `regression.rs`

rust Copy Edit

```rust
use linfa::prelude::*;
use linfa_logistic::LogisticRegression;
use ndarray::{Array2, ArrayBase, Axis, Ix2};
use linfa::dataset::{DatasetBase};

pub type Label = i32;

pub fn fit_logistic_model(
    data: &Array2<f32>,
    max_iter: u32,
) -> Result<LogisticRegression<f32>, Box<dyn std::error::Error>> {
    // extract features and labels here
}

pub struct Metrics {
    pub accuracy: f32,
    pub f1: f32,
    pub confusion: String,
}

pub fn evaluate_model(
    model: &LogisticRegression<f32>,
    test_data: &Array2<f32>,
) -> Result<Metrics, Box<dyn std::error::Error>> {
    // split test_data into X and y
    // make predictions
    // compute confusion matrix, accuracy, f1
}
```

3. Public API of `data_split.rs`

rust Copy Edit

```rust
/// Split an ndarray table into SmartCore `Dataset`s
pub fn train_test_from_ndarray(
    x: &Array2<f32>,          // features matrix
    y: &Array1<f32>,          // label vector
    test_ratio: f32,          // e.g. 0.2
    shuffle: bool,
    seed: Option<u64>,        // make runs reproducible
) -> Result<
        (Dataset<f32>, Dataset<f32>), // (train, test)
        Box<dyn std::error::Error>
    >;
```

Internals

1. **Convert** `x` (and `y`) to `DenseMatrix<f32>` / `Vec<f32>` using a thin helper.
2. Call `smartcore::model_selection::train_test_split`.
3. Wrap the four slices into two `Dataset::new` objects (SmartCore's convenience struct).
4. Return `Result<>` so callers can `?`-propagate errors.

4. Typical call-flow inside `main.rs`

rust Copy Edit

```rust
let path = "data/processed_titanic.csv";
let full = data_loader::load_csv(path, ',', true)?;      // ndarray

let (x, y) = utils::split_features_target(&full, target_idx); // if you add this helper
let (train_ds, test_ds) =
    data_split::train_test_from_ndarray(&x, &y, 0.3, true, Some(42))?;
println!("Train rows: {}, Test rows: {}", train_ds.num_samples(), test_ds.num_samples());
```

5. Helper additions (optional but handy)

- `utils.rs`
  - `fn split_features_target(table: &Array2<f32>, target_col: usize) -> (Array2<f32>, Array1<f32>)` – pulls the last (or arbitrary) column out as `y`.
  - A generic `fn ndarray_to_dense(a: &Array2<f32>) -> DenseMatrix<f32>`.
- **Logging / diagnostics** – reuse `print_head` from loader to preview each partition.

6. Testing & examples

- **Unit test in** `data_split.rs` that loads a tiny hard-coded matrix, splits 50-50, and asserts row counts + disjointness.
- **Integration test** in `tests/` that exercises the complete CSV-→split pipeline.