Detailed Project Plan: Constrained Portfolio Risk-Return Prediction

Aamer Jalan & Arin Baswana

Spring 2025

Contents

1 Introduction

This document outlines a comprehensive plan for building a Constrained Portfolio Risk-Return Prediction system using the Kaggle dataset:

https://www.kaggle.com/datasets/ehallmar/daily-historical-stock-prices-1970-2018

We will implement everything in **pure Python** with only `NumPy` and `pandas` (no additional ML libraries). Our main aims include:

- **Goal 1:** Demonstrate the ability to cleanse, transform, and engineer features from high-volume time-series stock data.
- **Goal 2:** Showcase custom implementations of linear and logistic regression (with L1/L2 regularization).
- **Goal 3:** Incorporate portfolio-weight constraints for risk-return optimization on multiple stocks.
- **Goal 4:** Prepare a foundation for advanced ML techniques: multi-class softmax classification, support vector machines (SVM), and kernel methods.
- **Goal 5:** Evaluate performance via regression, classification, and portfolio-specific metrics to illustrate a complete pipeline.

This approach demonstrates a firm understanding of each algorithmic concept and its real-world application in stock market analysis.

2 Project Directory Structure

```
portfolio-risk-return/

 data/
    raw/
       your_dataset.csv
    interim/
       cleaned_data.csv
    processed/
        features.csv

 notebooks/
    1_data_exploration.ipynb
    2_feature_exploration.ipynb
    3_preliminary_models.ipynb

 src/
```

```
    __init__.py
    config.py
    data_preprocessing.py
    feature_engineering.py
    constraints.py
    linear_regression.py
    logistic_regression.py
    portfolio_optimization.py
    train.py
    evaluate.py
    utils.py

main.py
requirements.txt
README.md
```

2.1 Description of Folders and Files (With Specific Goals)

- **data/**:
  - `data/raw/`: Original dataset(s) for historical stock prices, placed here to preserve integrity.
  - `data/interim/`: Intermediate outputs, such as partially cleaned data, for reproducible checkpoints.
  - `data/processed/`: Final feature-engineered data sets with all transforms and indicators, ready for modeling.
    - **Goal:** Ensure data transformations are traceable and consistent for iterative experiments.
- **notebooks/**: Jupyter notebooks to examine data quality and conduct quick experiments.
  - `1_data_exploration.ipynb`: Exploratory Data Analysis (EDA) and sanity checks.
  - `2_feature_exploration.ipynb`: Testing technical indicators (RSI, Bollinger, etc.) for correlation with future returns.
  - `3_preliminary_models.ipynb`: Rapid prototyping of the regression/classification approach.
    - **Goal:** Quickly identify which signals or engineered features might be most predictive for daily stock movement.
- **src/**: Core pipeline code.
  - `__init__.py`: Makes this directory a Python package.
  - `config.py`: Central configuration for hyperparameters, e.g. learning rate, regularization strength, portfolio constraints.

- `data_preprocessing.py`: Loads and cleans raw data (missing values, outliers, etc.).
  * **Goal:** Provide robust data cleaning tailored to large time-series, ensuring no date misalignment or invalid price.
- `feature_engineering.py`: Implements rolling mean, RSI, Bollinger Bands, and merges macro data if available.
  * **Goal:** Produce advanced, domain-relevant features for regression/classification.
- `constraints.py`: Functions for L1/L2 penalties and portfolio constraints (sum of weights = 1, max weight per stock, etc.).
  * **Goal:** Combine standard ML regularization with domain-specific constraints for portfolio risk management.
- `linear_regression.py`: Custom linear regression (gradient descent or normal equations), supporting L1/L2.
  * **Goal:** Predict next-day returns or prices, demonstrating an understanding of cost functions and optimization.
- `logistic_regression.py`: Binary logistic (up/down) and an extension path for multi-class (softmax).
  * **Goal:** Classify daily direction or discrete return bins, proving comprehension of classification basics.
- `portfolio_optimization.py`: Weight allocation logic, ensuring total weights sum to 1 and meeting constraints.
  * **Goal:** Demonstrate real-world application of combining forecasts with capital allocation constraints.
- `train.py`: High-level script to instantiate models, fit them, and optionally do immediate portfolio weighting.
  * **Goal:** Show the end-to-end process of data ingestion, training, and interim result storage.
- `evaluate.py`: Functions to compute MSE, accuracy, Sharpe ratio, etc.
  * **Goal:** Produce both ML metrics (for algorithm validation) and finance metrics (for portfolio performance).
- `utils.py`: Helper routines (metrics, logging, train-test splits, etc.) to avoid duplication.
  * **Goal:** Simplify the main scripts and centralize common procedures (e.g., time-series cross-validation).

• **main.py**:
  - Entry point that ties together data preprocessing, feature engineering, training, and evaluation.

  - **Goal:** Make the entire pipeline reproducible via a single `python main.py` command.

- **requirements.txt**: Minimal dependencies, specifying Python and known library versions.
- **README.md**: Installation steps, usage, and project overview.

3 Step-by-Step Implementation Plan

3.1 Data Acquisition and Setup

1. **Download Dataset:**
   - https://www.kaggle.com/datasets/ehallmar/daily-historical-stock-prices-1970-201
   - Place in `data/raw/` (e.g., `your_dataset.csv`).
   - **Goal:** Have a substantial historical timeseries with many tickers for multi-stock analyses.
2. **Environment Setup:**
   - Create a virtual environment (optional but recommended).
   - `pip install -r requirements.txt`.
   - **Goal:** Ensure reproducibility of environment for all team members and graders.

3.2 Data Preprocessing (`data_preprocessing.py`)

1. **Read Raw CSV:** Load `your_dataset.csv` into a `pandas` DataFrame.
2. **Handle Missing Values:**
   - Forward fill or a suitable imputation strategy.
   - Potentially drop extremely sparse data for defunct tickers.
   - **Goal:** Guarantee no breaks in timeseries alignment, especially for rolling calculations.
3. **Outlier Treatment:**
   - Winsorize or remove extreme outliers that could skew results heavily.
   - **Goal:** Maintain realistic price/volume data distribution without letting anomalies dominate.
4. **Normalization or Scaling:**
   - Optionally apply log transform to volumes or standardize price columns.
   - **Goal:** Improve numerical stability for gradient-based methods.
5. **Save to Interim:** `data/interim/cleaned_data.csv`.

3.3 Feature Engineering (`feature_engineering.py`)

1. **Read Cleaned Data:** Load `cleaned_data.csv`.
2. **Create Technical Indicators:**
   - Rolling mean/variance, Bollinger Bands, RSI, etc.
   - Daily returns: $\text{return}_t = \frac{\text{Close}_t - \text{Close}_{t-1}}{\text{Close}_{t-1}}$.
   - **Goal:** Capture short-term momentum, volatility, and price trends for each ticker.
3. **Add Macroeconomic Indicators (Optional):**
   - Merge inflation or interest rates on matching dates.
   - **Goal:** Account for broader economic context in price movements.
4. **Finalize Feature Matrix and Target:**
   - *Regression Target*: Next-day return or next-day price.
   - *Classification Target*: Direction (up/down) or multi-class (negative, neutral, positive).
   - **Goal:** Enable flexible forecasting tasks without changing the entire pipeline structure.
5. **Save to Processed:** `data/processed/features.csv`.

3.4 Constraints (`constraints.py`)

1. **L1 Penalty (LASSO):** $\lambda \sum_i |w_i|$.
2. **L2 Penalty (Ridge):** $\lambda \sum_i w_i^2$.
3. **Portfolio Constraints:**
   - $\sum_i w_i = 1$ (fully invested).
   - $|w_i| \leq \alpha$ for maximum position caps.
   - **Goal:** Combine standard ML regularization with real portfolio weight restrictions.

3.5 Linear Regression (`linear_regression.py`)

1. **Class Definition:** `class CustomLinearRegression:`
2. **Cost Function:**

$$J = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \; + \; \text{Regularization Terms (L1 or L2)}. \tag{1}$$

3. **Gradient Computation:**
   - Incorporate λw for L2 or sub-gradient for L1 into the update.
   - **Goal:** Show mastery of partial derivative derivations and optimization logic.
4. **Optimization:**
   - **Gradient Descent:** Iterate until convergence or max epochs.
   - **Normal Equations:** $(X^T X + \lambda I)w = X^T y$ (for ridge).
   - **Goal:** Illustrate an understanding of multiple solution methods.
5. **Predict Method:** $\hat{y} = Xw$.

3.6 Logistic Regression (`logistic_regression.py`) & Multi-class Softmax

1. **Binary Classification:**
$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad z = Xw. \tag{2}$$
2. **Cost Function (Binary Cross-Entropy):**
$$J = -\frac{1}{m} \sum_{i=1}^{m} \Big[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \Big] + \text{Regularization.} \tag{3}$$
3. **Gradient Updates:** Add L1 or L2 as needed.
4. **Multi-class Extension (Softmax):**
$$\hat{y}_{i,j} = \frac{e^{z_{i,j}}}{\sum_k e^{z_{i,k}}}. \tag{4}$$
5. **Multinomial Cross-Entropy:**
$$J = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} y_{i,j} \log(\hat{y}_{i,j}). \tag{5}$$
6. **Goal:** Classify daily price movement direction or label returns in multiple "bins" (e.g., down, flat, up).

3.7 Portfolio Optimization (`portfolio_optimization.py`)

1. **Input:** Model-predicted returns (regression) or upward probabilities (classification).
2. **Constraint Methods:**
   - $\sum_i w_i = 1$.
   - $|w_i| \leq \alpha$, e.g., 10% max position.
   - **Goal:** Convert predictive signals into real investment decisions under strict constraints.
3. **Risk Measures:**
   - Optionally incorporate covariance-based portfolio variance minimization for advanced risk control.
4. **Solver Implementation:**
   - Could do direct quadratic programming in NumPy for minimal variance or risk-adjusted returns.
   - **Goal:** Show how theoretical optimization integrates with stock forecasting.

3.8 Training Pipeline (`train.py`)

1. **Load Data:** `features.csv` from `data/processed/`.
2. **Split:** Train/Validation/Test or rolling-window backtest segments.
3. **Initialize Model:** For example, `CustomLinearRegression(penalty='l2', lambda_val=0.1)`.
4. **Fit Model:** `model.fit(X_train, y_train)`.
5. **Optional: Portfolio Weights:** If needed, call `portfolio_optimization.py` with predicted returns.
6. **Save Results:** Store learned weights, predictions, or portfolio allocations.
7. **Goal:** Show entire pipeline from raw data to final model outputs is integrated in one script.

3.9 Evaluation (`evaluate.py`)

1. **Regression Metrics:**
   - MSE, RMSE, MAE
   - R^2
   - **Goal:** Measure how closely predicted returns/prices match reality.

2. **Classification Metrics:**
   - Accuracy, Precision, Recall, F1
   - Confusion Matrix
   - **Goal:** For directional or multi-class tasks, judge classification quality.
3. **Portfolio Metrics:**
   - Mean Return, Volatility, Sharpe Ratio
   - Max Drawdown (advanced)
   - **Goal:** Assess real-world feasibility of the strategy, going beyond typical ML metrics.

3.10 High-Level Script (`main.py`)

1. **Orchestrate Entire Workflow:**
   - `clean_data()` from `data_preprocessing`.
   - `generate_features()` from `feature_engineering`.
   - Run `train_main()` from `train.py`.
   - Execute `evaluate_model()` from `evaluate.py`.
   - **Goal:** Provide a single command to reproduce the entire pipeline, from raw data to final metrics.

4 Suggestions for Advanced Extensions

Below are additional methods to deepen your project and highlight broader ML capabilities:

4.1 Support Vector Machines (SVM)

- **Custom Implementation:** Implement hinge-loss minimization using primal or dual form with gradient-based methods in pure Python.
- **Kernel Methods:** Extend to RBF, polynomial, or other kernels for capturing non-linear relationships.
- **Use Case:** Classify daily direction more robustly (especially in multi-class volatility regimes).
- **Goal:** Demonstrate knowledge of maximum margin classifiers and kernel expansions in a time-series context.

4.2 Multi-class Softmax

- **Discrete Return Bins:** For example, strongly negative, mildly negative, neutral, mildly positive, strongly positive.
- **Implementation:** Extend binary logistic to K-class softmax with cross-entropy loss.
- **Goal:** Show advanced classification usage with real financial data distributions.

4.3 Advanced Portfolio Constraints

- **Factor Exposure**: e.g., limiting sector or factor weights (like Fama-French factors).
- **Transaction Costs**: Incorporate a penalty for frequent rebalancing or large trades.
- **Goal:** Make the portfolio optimization more realistic and academically thorough.

4.4 Additional Regularization or Optimization Techniques

- **Elastic Net:** Combine L1 (sparseness) and L2 (stability).
- **Coordinate Descent or Proximal Gradient:** Efficient for L1 and L1+L2 penalties.
- **Goal:** Show advanced iterative optimization methods for large-scale feature sets.

4.5 Model Ensembling and Stacking

- **Multiple Custom Models:** Combine logistic, linear, SVM predictions.
- **Stacking Mechanisms:** Train a meta-model on top of outputs from base models.
- **Goal:** Illustrate ensemble improvements without external libraries, purely from scratch.

4.6 Time-Series Cross-Validation

- **Rolling Windows:** Split historically, train on a window, test on the next period, move forward.
- **Walk-Forward Analysis:** Frequent re-training to adapt to new market conditions.
- **Goal:** Provide a robust evaluation that simulates how actual trading strategies behave over time.

5 Conclusion

This plan provides a thorough, methodical framework for a portfolio-focused machine learning project using historical stock data from 1970–2018. By progressively adding linear/logistic regression, multi-class classification, SVMs, and portfolio constraints, you demonstrate a strong command of **both theoretical ML concepts** and their **practical, domain-specific application** in finance. The chosen file structure and step-wise breakdown ensures clarity, modularity, and scalability, making it straightforward to enrich the project with more advanced techniques or more complex financial constraints.