Backtesting Portfolio Weight Optimization

Dr. Sebastian Stöckl

2024-11-28

Introduction

Modern portfolio optimization requires balancing risk-adjusted returns with real-world constraints, such as transaction costs, leverage limits, and benchmark adherence. Traditional optimization techniques often struggle to incorporate these constraints effectively, leaving room for innovative approaches.

This part of the **InvestigatoR** package leverages machine learning, particularly Keras-based models, to address these challenges. By offering flexible customization of loss functions, penalties, and activations, it enables users to design portfolios that maximize performance while adhering to practical constraints.

This vignette serves to:

1. Introduce the InvestigatoR package's capabilities for portfolio optimization.
2. Demonstrate its functionality through backtesting.
3. Validate its methods empirically, showcasing improvements in portfolio performance metrics like the Sharpe Ratio and Information Ratio.

Leveraging Neural Networks for Portfolio Weight Prediction

Portfolio weight prediction in the **InvestigatoR** package is achieved using the `keras_weights` function, which leverages the power of TensorFlow and Keras to train neural networks. This approach provides:

- **Flexibility**: Easy customization of loss functions, activation functions, and metrics.
- **Scalability**: Ability to handle large datasets and numerous features efficiently.
- **Extensibility**: Support for diverse portfolio optimization objectives through modular loss functions.

Loss Functions for `keras_weights`

The `keras_weights` function leverages the flexibility of **Keras** and **TensorFlow** to enable weight prediction for portfolio optimization tasks. It supports highly customizable configurations through user-defined layers, activation functions, loss functions, optimizers, and metrics. The modularity of this setup allows users to experiment with various optimization objectives, including Sharpe ratio, Information ratio, and active returns, while incorporating constraints like L1/L2 regularization, leverage, and diversification penalties.

Available Loss Functions and Parameters

1. `sharpe_ratio_loss`
   - **Objective**: Maximizes the Sharpe Ratio of portfolio returns.

- **Parameters**:
    - `transaction_costs`: Penalizes frequent rebalancing to reduce costs.
    - `delta`: Tolerance level for achieving the optimal Sharpe Ratio.
    - `lambda`: Diversification penalty, encouraging broader allocation.
    - `leverage`: Constraint on total portfolio exposure.
    - `eta`: Leverage penalty, discouraging extreme allocations.

2. `sharpe_ratio_difference_loss`
    - **Objective**: Maximizes the Sharpe Ratio of active returns (portfolio return minus benchmark return).
    - **Parameters**:
        - Same as `sharpe_ratio_loss`, with additional `benchmark_label` for specifying the benchmark.
3. `information_ratio_loss_active_returns`
    - **Objective**: Maximizes the Information Ratio, balancing active returns and tracking error.
    - **Parameters**:
        - `lambda_l1`: Encourages sparsity in active weight changes.
        - `lambda_l2`: Promotes smooth weight deviations.
4. `information_ratio_loss_regression_based`
    - **Objective**: Maximizes the Information Ratio, based on regression alpha and idiosyncratic volatility.
    - **Parameters**:
        - `lambda_l1`: Encourages sparsity in active weight changes.
        - `lambda_l2`: Promotes smooth weight deviations.
5. **Custom Loss Functions**
    - Users can define their loss functions by specifying objectives and penalties, integrating seamlessly with `keras_weights`.

Metrics and Activation Functions for `keras_weights`

Metrics provide additional performance measures during training, complementing the loss functions. Examples include:

- Portfolio return metrics (e.g., cumulative return, annualized return).
- Risk-adjusted performance metrics (e.g., Sharpe Ratio, Information Ratio).
- Constraints validation metrics (e.g., turnover, leverage).

Metrics can be customized to align with specific portfolio objectives, offering valuable insights during model training.

Activation functions in `keras_weights` control the characteristics of portfolio weights, ensuring adherence to practical constraints.

Available Activation Functions

1. `activation_box_sigmoid`
    - **Objective**: Constrain weights to a specific range (e.g., [0, 0.2]).
    - **Parameters**:
        - `min_weight`: Lower bound for weights.
        - `max_weight`: Upper bound for weights.
2. `activation_box_tanh`
    - **Objective**: Constrain weights within a symmetric range (e.g., [-0.1, 0.1]).
    - **Parameters**:
        - `min_weight`: Lower bound for weights.
        - `max_weight`: Upper bound for weights.

These activation functions allow the user to impose realistic constraints on portfolio allocations, ensuring practical applicability. Also, self-provided activation functions can be used.

Objectives

The following sections demonstrate: 1. **Direct Portfolio Optimization**: Training neural networks to directly optimize weights without a benchmark. 2. **Benchmark-Relative Optimization**: Enhancing portfolio performance relative to a benchmark, leveraging advanced loss functions and constraints.

Backtesting Framework Overview

The **backtesting_weights** function provides a robust and flexible framework for predicting portfolio weights using various models and evaluating their performance. Here's a detailed explanation of its components, parameters, and best practices for configuration.

Key Components of the Framework

1. **Data Input**:
    - **Format**: The data must be in a long format with columns for `stock_id`, `date`, feature variables, actual returns, benchmark weights (if any), and an optional mask.
    - **Features**: The user specifies which feature columns will be used by the model for weight predictions.
2. **Prediction Models**:
    - Multiple models can be defined in the `pf_config` parameter, each with one or more configurations.
    - **Model Specification**:

    - A `weight_func` key defines the function used to predict weights.
    - Each configuration can specify constraints (e.g., min/max weights) or other hyperparameters.
  - Models are processed sequentially, and their outputs (predicted weights) are stored in the `portfolio_object`.
3. **Rolling Window Backtesting**:
  - **Purpose**: Ensures the model uses only past data for predictions, avoiding look-ahead bias.
  - **Parameters**:
    - `window_size`: Specifies the training period (e.g., “5 years”).
    - `step_size`: Defines the interval between successive training windows (e.g., “1 month”).
    - `offset`: Adds a buffer to avoid overlap or data leakage (e.g., “1 month”).
  - **In-Sample (IS) Testing**:
    - Optionally adds an in-sample evaluation phase to analyze model fit on the training data.
4. **Parallel Processing**:
  - Utilizes multiple cores for faster computation when `num_cores` is specified.
  - The `furrr` package enables parallel execution of predictions.
5. **Postprocessing**:
  - Predicted weights are appended to the `portfolio_object`.
  - Additional postprocessing (e.g., normalization, turnover constraints) is performed later.

Key Parameters and Their Roles

1. **Data and Features**:
  - `data`: Input dataset with all necessary variables.
  - `return_label`: Column name for actual returns, used to calculate portfolio performance.
  - `features`: List of predictor variables for weight prediction.
2. **Benchmarking**:
  - `benchmark_label`: Optional column name for benchmark weights.
  - Allows calculation of metrics like tracking error, information ratio, and active share.
3. **Backtesting Settings**:
  - `rolling`: Boolean flag indicating whether to use a rolling window approach.
  - `window_size`: Defines the training period (e.g., “5 years”).
  - `step_size`: Specifies the interval between consecutive predictions (e.g., “1 month”).
  - `offset`: Ensures predictions don’t use overlapping data.
4. **Model Configurations**:
  - `pf_config`: List of models and configurations, each specifying:
    - Prediction function (`weight_func`).
    - Constraints like `min_weight` and `max_weight`.
    - Other model-specific parameters (e.g., regularization terms).
5. **Performance Enhancements**:
  - `num_cores`: Enables parallel processing for faster backtesting.
  - `verbose`: Provides progress updates and diagnostic information.

Workflow

1. **Input Validation**:
  - Ensures the data and parameters meet the required format and constraints.
2. **Rolling Window Setup**:
  - Defines training and prediction periods based on `window_size`, `step_size`, and `offset`.
3. **Weight Prediction**:
  - For each model and configuration:
    - A weight prediction function (`weight_func`) is applied to the training data.
    - Predicted weights are stored with associated metadata.
4. **Portfolio Construction**:
  - Predicted weights are appended to the `portfolio_object`.
  - Subsequent postprocessing handles constraints (e.g., turnover, leverage).
5. **Performance Evaluation**:
  - Metrics like Sharpe Ratio, Sortino Ratio, active share, and turnover are calculated.
  - Advanced metrics (e.g., conditional VaR) can also be included.

Best Practices for Configuration

1. **Data Preparation**:
  - Ensure the data is complete and free of missing values.
  - Use a consistent frequency (e.g., monthly, daily) for `date`.
2. **Model Selection**:
  - Use separate configurations to test different hyperparameters or constraints.
3. **Rolling Window Design**:
  - Use a sufficiently large `window_size` (e.g., 3–5 years) to capture meaningful trends.
  - Choose a `step_size` that balances granularity and computational cost (e.g., “1 year”).
4. **Constraints**:
  - **Weight Limits**: Use `min_weight` and `max_weight` to enforce realistic portfolio allocations, first through the activation function, second through postprocessing.

- **Turnover**: Monitor and control trading activity to minimize transaction costs.

5. **Parallelization**:
    - Enable `num_cores` for large datasets or computationally intensive models.

Output

The resulting `portfolioReturns` object contains:

- **Predicted Weights**: Allocations for each stock over time.
- **Portfolio Returns**: Periodic returns based on predicted weights.
- **Benchmark Weights**: If required/given, benchmark allocations.
- **Benchmark Returns**: If required/given, benchmark returns.
- **Configurations**: All specs applied during portfolio optimization and postprocessing.

In the following we will step by step introduce the various setups, while simultaneously elaborating on their implications and results.

Setup

Scientific Context and Requirements

The examples presented here are computationally intensive, as they involve training machine learning models and running backtests on financial data. We aim to demonstrate the process of directly optimizing a portfolio and optimizing a portfolio relative to its benchmark. We utilize the dataset provided by Guillaume Coqueret, which accompanies his book “Machine Learning for Factor Investing” available at mlfactor.com.

Dataset Description

Ac cordingv to thge website, the dataset comprises information on 1,207 stocks listed in the U.S., including firms from Canada and Mexico. It spans from November 1998 to March 2019 and includes 93 firm-specific attributes per time point. These attributes cover various aspects such as valuation (e.g., earning yields, accounting ratios), profitability and quality (e.g., return on equity), momentum and technical analysis (e.g., past returns, relative strength index), risk (e.g., volatilities), estimates (e.g., earnings-per-share), and volume and liquidity (e.g., share turnover). The dataset is not perfectly rectangular; the number of firms and their attributes vary over time, reflecting a realistic financial environment.

ibraries and Environment Setup

```
# Load necessary libraries
library(reticulate)
library(tensorflow)
library(keras3)
library(dplyr)
library(tibble)
library(PerformanceAnalytics)

# Set up the Python virtual environment
reticulate::use_virtualenv("../../python/")

# Confirm TensorFlow and Keras availability
reticulate::py_config()
```

```
## python:         /home/sstoeckl/Packages/python/bin/python
## libpython:      /usr/lib/python3.10/config-3.10-x86_64-linux-gnu/libpython3.10.so
## pythonhome:     /home/sstoeckl/Packages/python:/home/sstoeckl/Packages/python
## version:        3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
## numpy:          /home/sstoeckl/Packages/python/lib/python3.10/site-packages/numpy
## numpy_version:  1.26.4
## keras:          /home/sstoeckl/Packages/python/lib/python3.10/site-packages/keras
##
## NOTE: Python version was forced by use_python() function
```

```
reticulate::py_module_available("tensorflow")
```

```
## [1] TRUE
```

```
reticulate::py_module_available("keras")
```

```
## [1] TRUE
```

```
# Load InvestigatoR package
# library(InvestigatoR)
devtools::load_all()
```

Data Preparation

We now read the data and quickly summarise its poroperties.

```r
# Load the dataset
data("data_ml")

# Summarize the dataset
data_ml %>%
  summarise(
    n_assets = n_distinct(stock_id),
    start_date = min(date),
    end_date = max(date)
  )
```

```
## # A tibble: 1 × 3
##   n_assets start_date end_date
##      <int> <date>     <date>
## 1     1207 1998-11-30 2019-03-31
```

Filtering the Data for Backtesting

To streamline computations, we will:

1. Restrict the dataset to the 100 assets with the most complete data.
2. Limit the time period to 2000-01-01 through 2015-12-31.

```r
# Identify assets with complete data for the full time period
stock_ids <- data_ml %>%
  group_by(stock_id) %>%
  summarise(
    start_date = min(date),
    end_date = max(date),
    n_obs = n()
  ) %>%
  filter(n_obs == max(n_obs)) %>%
  distinct(stock_id) #%>%
  #dplyr::slice(1:100)  # Limit to 100 assets for speed

# Filter the dataset to a reduced subset
data_ml_red <- data_ml %>%
  right_join(stock_ids, by = "stock_id") %>%
  filter(date >= "2000-01-01", date <= "2015-12-31")
```

Summarizing the Filtered Dataset

Before proceeding, we summarize the reduced dataset to verify it is ready for backtesting.

```r
# Summarize filtered dataset
data_ml_red %>%
  summarise(
    n_assets = n_distinct(stock_id),
    n_periods = n_distinct(date),
    start_date = min(date),
    end_date = max(date)
  )
```

```
## # A tibble: 1 × 4
##   n_assets n_periods start_date end_date
##      <int>     <int> <date>     <date>
## 1      372       192 2000-01-31 2015-12-31
```

This structured setup provides a solid foundation for the backtesting analysis that follows. In the next sections, we will:

- Configure machine learning models for portfolio optimization.
- Evaluate standalone and benchmark-tilted optimization techniques.
- Empirically validate their performance through metrics like the Sharpe Ratio and Information Ratio.

Standalone Portfolio Optimization

In the world of portfolio management, balancing **risk** and **return** is one of the most critical and challenging objectives. Traditional methods often involve forecasting expected returns for individual assets and then using these forecasts to determine portfolio weights. However, this two-step process introduces several layers of uncertainty, including the challenge of accurately predicting future returns and the potential propagation of forecast errors into portfolio construction.

Standalone portfolio optimization takes a fundamentally different approach. Instead of predicting returns first and then determining weights, this method directly optimizes portfolio weights to achieve a specific objective, such as maximizing the **Sharpe ratio** or optimizing another risk-return tradeoff function like **quadratic utility**.

Why Optimize Portfolio Weights Directly?

1. **Avoiding Forecasting Errors**:
    - Predicting returns is notoriously difficult and subject to substantial noise and biases.
    - By directly optimizing portfolio weights, this approach eliminates the dependence on return predictions, focusing instead on achieving a predefined portfolio-level goal.
2. **Customizable Objectives**:
    - The framework can accommodate different objectives beyond the Sharpe ratio, such as:
        - **Quadratic Utility Maximization**: Balancing expected returns against risk based on investor risk aversion.
        - **Risk Minimization**: Focusing solely on reducing portfolio volatility.
        - **ESG Optimization**: Incorporating sustainability constraints and objectives.
    - This flexibility makes the method adaptable to a wide range of investment mandates.
3. **Improved Practicality**:
    - Direct weight optimization often leads to more stable and actionable portfolios by incorporating practical constraints (e.g., weight bounds, turnover limits) during the optimization process.

The Role of the Sharpe Ratio

The Sharpe ratio is a cornerstone of modern portfolio optimization and measures the portfolio's return per unit of risk. Mathematically, it is defined as:

$$S = \frac{\mathbb{E}[R_p - R_f]}{\sigma_p}$$

Where:

- R_p: Portfolio return.
- R_f: Risk-free rate.
- σ_p: Portfolio volatility.

Maximizing the Sharpe ratio ensures that the portfolio achieves the highest possible return for a given level of risk. It reflects the fundamental tradeoff in investing: taking on additional risk should lead to proportionally higher returns. However, maximizing the Sharpe ratio is not without challenges:

1. **Estimation Errors**:
    - Estimating the covariance matrix of asset returns accurately is difficult, particularly in high-dimensional settings or with limited data.
    - Errors in estimation can lead to suboptimal or unstable portfolios.
2. **Nonlinear Optimization**:
    - The Sharpe ratio introduces nonlinearity in the optimization problem, making the computation more complex and prone to numerical instability.

Despite these challenges, the Sharpe ratio remains a widely used and intuitive objective for portfolio optimization.

Alternative Optimization Objectives

While the Sharpe ratio is a popular choice, other objective functions are worth considering:

1. **Quadratic Utility Maximization**:
    - Incorporates an investor's risk aversion (γ), optimizing for the tradeoff between expected returns and risk:

$$U = \mathbb{E}[R_p] - \frac{\gamma}{2}\sigma_p^2$$

    - Investors with higher risk aversion prioritize minimizing volatility, while those with lower risk aversion seek higher returns.
2. **Minimum Variance Portfolios**:
    - Focuses solely on minimizing risk (σ_p^2) without considering expected returns.
    - Useful for highly risk-averse investors or in environments with high return uncertainty.
3. **Other Custom Objectives**:
    - ESG or thematic constraints (e.g., carbon footprint minimization).
    - Tail risk measures like Value at Risk (VaR) or Conditional VaR (CVaR).

The Benchmark: The Equally Weighted Portfolio

One of the toughest benchmarks to beat in practice is the **equally weighted portfolio (EWP)**, where each asset is assigned the same weight. Despite its simplicity, the EWP has several attractive properties:

1. **Diversification**:
    - Equal weights ensure no single asset dominates the portfolio, reducing concentration risk.
2. **Robustness**:
    - Unlike optimized portfolios, the EWP does not rely on noisy estimates of returns or covariances, making it less prone to estimation errors.
3. **Simplicity**:
    - The EWP is easy to implement and interpret, making it an appealing benchmark for investors and researchers alike.

Beating the EWP consistently is challenging because it implicitly harnesses the benefits of diversification while avoiding estimation risk. Any optimization approach must demonstrate clear and consistent advantages over this deceptively simple benchmark.

Balancing Risk and Return in Practice

Balancing risk and return is inherently difficult due to:

- **Uncertainty in Inputs**: Estimates of asset returns, volatilities, and correlations are often noisy and unstable.
- **Dynamic Market Conditions**: Asset behavior changes over time, requiring adaptive models.
- **Practical Constraints**: Real-world portfolios must account for turnover costs, leverage limits, and regulatory requirements, which complicate the optimization process.

Standalone portfolio optimization seeks to address these challenges by directly optimizing portfolio weights to align with investor objectives, subject to constraints that reflect real-world considerations.

Outline of the Following Sections

In the subsequent sections, we will:

1. **Specify Various Optimization Setups**:
    - Define different objective functions and constraints, such as turnover limits, leverage constraints, and weight bounds.
    - Discuss their theoretical motivations and practical implications.
2. **Elaborate on Their Benefits**:
    - Highlight the advantages of each setup, such as stability, efficiency, or adaptability to specific investment goals.
3. **Evaluate the Produced Weights**:
    - Assess how well the optimized weights align with the intended objectives.
    - Analyze the resulting portfolios' diversification and risk characteristics.
4. **Evaluate Portfolio Performance**:
    - Compare the performance of optimized portfolios against benchmarks like the EWP.
    - Use metrics such as the Sharpe ratio, turnover, active share, and drawdown to provide a comprehensive evaluation.

Standalone portfolio optimization provides a unique and powerful approach to portfolio construction, offering significant advantages over traditional methods. By directly targeting portfolio-level objectives, this method enables investors to navigate the complexities of risk and return more effectively.

Portfolio optimization without constraints often leads to unrealistic allocations, such as extreme leverage or high concentration in specific assets. In this section, we:

1. Configure a simple Keras model to optimize portfolio weights without constraints.
2. Backtest the portfolio and evaluate its performance using `summary.performance` and `summary.performance2`.
3. Use `summary.weights` to highlight unreasonable weight distributions.
4. Apply postprocessing to enforce practical constraints and reassess performance.

Unconstrained optimization

We begin by setting up a incorrectly specified model, which will be used to optimize the portfolio weights. This model will not enforce any constraints, leading to unrealistic allocations (not entirely, as the used output activation will enforce weights between 0 and 1). More specifically, we define a set of portfolio constraints for backtesting:

- **Long-Only**: No short-selling; weights must be non-negative.
- **Weight Constraints**: Individual weights must not exceed 10%.
- **Leverage Limit**: Total exposure is capped at 1.0 (fully invested).
- **Diversification Penalty**: Avoid over-concentration in a few assets.

These constraints will be applied in later sections, but here we start with an unconstrained optimization.

Model Configuration

We start with a simple Keras model with:

- Three dense layers (32, 16, and 1 unit respectively).
- *eLU* activation for hidden layers and *tanh* activation for the output layer (tanh is bounded between -1 and 1, which we will include, rather than the sigmoid function, for educational purposes).
- Loss function: `sharpe_ratio_loss` (no penalties applied).
- Adam optimizer with a learning rate of 0.001.
- 50 training epochs (for output reasons, the batch size is automatically set to be the entire training dataset, hence we need a number of epochs to achieve convergence).
- for reproducibility, we set the random seed to 42 (the answer to everything).

```
# Define a simple Keras model configuration
config_keras_simple <- list(
  layers = list(
    list(type = "dense", units =32, activation = "elu"),
    list(type = "dense", units = 16, activation = "elu"),
    list(type = "dense", units = 1, activation = "tanh")  # Simple tanh activation bounded between -1
        and 1
  ),
  loss = list(
    name = "sharpe_ratio_loss",  # Standard Sharpe Ratio loss with no penalties
    transaction_costs = 0,  # 5 basis points transaction costs
    delta = 0,  # Tolerance level for optimal Sharpe Ratio
```

```
    lambda = 0,  # Diversification penalty
    leverage = 0,  # Leverage limit
    eta = 0  # Leverage penalty
  ),
  optimizer = list(name = "optimizer_adam", learning_rate = 0.001),
  epochs = 50,
  verbose = 0,
  seeds = c(42)  # Set random seed for reproducibility
)
```

Backtesting the Unconstrained Portfolio

We backtest the portfolio using the unconstrained model. This involves:

- Training the model on a rolling 5 year window of historical data.
- Predicting portfolio weights for one year into the future.
- Including a one-month offset between training and prediction data to prevent data leakage.

```
# Define the portfolio configuration for all subsequent backtests
return_label <- "R1M_Usd"
features <- colnames(data_ml_red)[4:95]
rolling <- TRUE
window_size <- "5 years"
step_size <- "1 year"
offset <- "1 month"
in_sample <- TRUE
num_cores <- 12
verbose <- FALSE
```

Now we run the backtest:

```
# Run the backtest
portfolios_raw_unconstrained <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  features = features,
  pf_config = list(keras_weights_simple = list(weight_func = "keras_weights",config1 =
         config_keras_simple)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  in_sample = in_sample,
  num_cores = num_cores,
  verbose = verbose
)
```

Result Analysis

Unconstrained optimization often results in extreme weights. We use `summary.weights` to demonstrate this issue.

```
# Summarize portfolio weights
tt <- summary.weights(portfolios_raw_unconstrained, print=TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_simple_1	-0.9517/0.9914	0.1161 (0.477)	0.4374 (0.0732)	89.6652 (36.654)	0.012	155.9253 (34.033)	9.3429 (1.5413)	157.651 (45.2936)	31.0624	5963.988

Observation: The weight summary reveals:

- Large weight differences (Min/Max: -0.9517/0.9914, indicating extreme allocations, the timeseries average weight standard deviation (and its timeseries standard deviation) per date is 0.4374 (0.0732).
- Large total turnover at 5963.9883, indicating that throughout the backtest, the portfolio was completely turned over 5964 times.
- Concentration in a few assets, leading to poor diversification, based on the average Herfindahl-Hirschman Index (HHI) of 89.6652 (36.654).

We can evaluate the portfolio's performance using the `summary.performance` and/or `summary.performance2` functions. These provide insights into key metrics like Sharpe Ratio, turnover, and diversification.

```
# Summarize portfolio performance
tt <- summary.performance2(portfolios_raw_unconstrained, transaction_cost = 0.005, gamma = 3,
         test=TRUE, print=TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_simple_1	11.9614***	0.8787	77.8873	5963.9883	0.7435	0.0729***	-266.0084
benchmark	0.1533	0.7955	0.0000	16.4426	0.769	NaN	0.0976

Observation: Performance-wise we observe the following:

- The Sharpe Ratio is high at 0.8787, whereas the one of the equally weighted portfolios is 0.7955.
- The turnover, however, is very high at 5964, indicating excessive trading activity relative to the equally weighted benchmark at 16
- Taking into account transaction cost at 0.005 (5 basis points), this leads to a turnover-adjusted Sharpe Ratio of 0.7435, which is still above the equally weighted benchmark at 0.769.
- Due to the significantly larger annualized return of 11.9614*** compared to the benchmark at 0.1533, the certainty equivalent return is very negative at all levels of risk aversion (-266.0084), compared to the benchmark at 0.0976.

While the Sharpe Ratio may appear high, other metrics like turnover and diversification are very sub-optimal, making the portfolio impractical for real-world use.

Postprocessing the Portfolio

We could try post-optimization enforcement of constraints to improve the portfolio's practicality. Here, we apply a postprocessing configuration to enforce constraints on the portfolio weights:

```
# Define postprocessing configuration
pp_config <- list(
  list(operation = "set_weightsum",
       min_weight = 0, max_weight = 0.1, # box constraint on individual assets
       min_sum = 0.8, max_sum=1.5, # we do not yet include a full-investm,ent constraint without
         leverage
       allow_short_sale = FALSE))

# Apply postprocessing
portfolios_postprocessed <- postprocessing_portfolios(portfolios_raw_unconstrained, pp_config)

# Summarize postprocessed weights
tt <- summary.weights(portfolios_postprocessed, print=TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_simple_1	0/0.0102	0.004 (0.0038)	0.0036 (0.001)	0.0113 (0.0019)	91.8045	1.5 (0)	0.1059 (0.0095)	0 (0)	0.2892	55.5298

Observation: The postprocessing step enforces constraints on the portfolio weights, leading to:

- Somehow improved weights, which are now between in the interval 0/0.0102.
- The total turnover is reduced to 55.5298, indicating a more stable portfolio.
- The average Herfindahl-Hirschman Index (HHI) is now 0.0113 (0.0019), reflecting much better diversification.

Potentially this will make the portfolio be very close to the equally weighted portfolio, but with a slightly higher Sharpe Ratio.

```
# Evaluate performance after postprocessing
tt <- summary.performance2(portfolios_postprocessed, transaction_cost = 0.005, gamma = 5, test=TRUE,
        print=TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=5)
keras_weights_simple_1	0.291**	0.8973**	0.6848	55.5298	0.845	0.0818***	0.0280
benchmark	0.1533	0.7955	0.0000	16.4426	0.769	NaN	0.0604

Observation: The postprocessed portfolio shows:

- A much better Sharpe ratio, significantly exceeding the equally weighted portfolio at 0.8973** vs. 0.7955.
- This even holds when applying transaction cost (0.845 vs. 0.769), due to a much decreased turnover of 56.
- The certainty equivalent return is now positive at the given risk level, indicating a more stable and practical portfolio.

Results and Insights

1. **Before Postprocessing**:
    - The unconstrained portfolio exhibited extreme weights, high turnover, and poor diversification.
    - While achieving a high Sharpe Ratio, the portfolio's impracticality limits its usefulness.
2. **After Postprocessing**:

- Weight distributions improved significantly, adhering to realistic constraints.
- Diversification and turnover metrics became more acceptable, making the portfolio closer to being usable for real-world implementation.

Backtesting the Weight-constrained portfolio

Activation functions in the **InvestigatoR** package allow users to impose practical constraints on portfolio weights. Therefore, we can enforce constraints like non-negativity, bounded weights, and smooth transitions using activation functions like `activation_box_sigmoid` and `activation_box_tanh`.

The `activation_box_sigmoid` ensures:

- Non-negative weights (long-only).
- Bounded weights, e.g., within [0, 0.1].
- Smooth transitions at the boundaries to avoid discontinuities.

```
# Configuration with long-only activation
config_box_sigmoid <- config_keras_simple
config_box_sigmoid$layers[[3]]$activation <- activation_box_sigmoid(min_weight = 0, max_weight = 0.1)

# Backtest with box-sigmoid activation
portfolios_box_sigmoid <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  features = features,
  pf_config = list(keras_weights_box_sigmoid=list(weight_func = "keras_weights",config1 =
         config_box_sigmoid)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Result Analysis

To compare the results with the constrained cases, we use the `combine_portfolios` function to aggregate the portfolios.

```
# Combine portfolios for comparison
combined_portfolios <- combine_portfolioReturns(list(
  portfolios_postprocessed,
  portfolios_box_sigmoid,
  postprocessing_portfolios(portfolios_box_sigmoid, pp_config)
))

# Summarize combined weights
tt <- summary.weights(combined_portfolios, print=TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_simple_1	0/0.0102	0.004 (0.0038)	0.0036 (0.001)	0.0113 (0.0019)	91.8045	1.5 (0)	0.1059 (0.0095)	0 (0)	0.2892	55.5298
keras_weights_box_sigmoid_2	0/0.0684	0.0016 (0.0043)	0.0038 (0.0017)	0.0077 (0.0064)	492.3355	0.5942 (0.4039)	0.0797 (0.0366)	0 (0)	0.1599	30.7079
keras_weights_box_sigmoid_3	0/0.1	0.0024 (0.0071)	0.0068 (0.0021)	0.0209 (0.0102)	65.1258	0.8755 (0.1718)	0.1396 (0.0371)	0 (0)	0.2426	46.5831

Observation: The weight summary shows:

- The `activation_box_sigmoid` function naturally enforces bounded weights, leading to more realistic allocations. The min/max weights are now 0/0.0684 (and 0/0.1 after postprocessing).
- The average weight standard deviation per date is 0.0038 (0.0017), indicating a very stable distribution.
- The average Herfindahl-Hirschman Index (HHI) is now very low at 0.0077 (0.0064) (and 0.0209 (0.0102) after postprocessing), reflecting excellent diversification that is slightly scaled up after postprocessing.
- The total turnover is significantly reduced to 30.7079 (and 46.5831 after postprocessing).

```
# Summarize combined performance
tt<- summary.performance2(combined_portfolios, transaction_cost = 0.005, gamma = 3, test=TRUE,
         print=TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_simple_1	0.291**	0.8973**	0.6848	55.5298	0.845	0.0818***	0.1332
keras_weights_box_sigmoid_2	0.1641	1.2357**	0.5310	30.7079	1.1697**	0.0065	0.1376

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_box_sigmoid_3	0.2459**	1.2137***	0.6188	46.5831	1.1451***	0.0726***	0.1843
benchmark	0.1533	0.7955	0.0000	16.4426	0.769	NaN	0.0976

Observation: The performance summary reveals:

- The annualized Sharpe Ratio of the postprocessed portfolio is 1.2137***, significantly exceeding the equally weighted benchmark at 0.7955 and also exceeding the one of the (postprocessed) unconstrained optimization at 0.8973**.
- Due to the much smaller overall turnover, the turnover-adjusted Sharpe Ratio of the postprocessed portfolio is 1.1451***, the highest among all of those portfolios tested so far!

Results and Insights

1. **Constrained Portfolio Optimization**:
   - The `activation_box_sigmoid` function enforces practical constraints on portfolio weights.
   - Improved diversification and reduced turnover compared to unconstrained portfolios.
2. **Performance Metrics**:
   - The constrained portfolio exhibits a higher Sharpe Ratio and lower turnover, indicating better risk-return characteristics.
   - The portfolio outperforms the equally weighted benchmark and the unconstrained portfolio, especially after postprocessing.

These results highlight the importance of activation functions in creating portfolios that adhere to practical constraints while maintaining robust performance. In the next section, we will penalize the loss function with various parameters and observe their impact.

Constraining `sharpe_ratio_loss`

Portfolio optimization often requires balancing return and risk while adhering to practical constraints. The `sharpe_ratio_loss` function in the **InvestigatoR** package provides the flexibility to incorporate various penalties, enabling portfolios to reflect real-world requirements.

Key Parameters in `sharpe_ratio_loss`

1. **Transaction Costs (`transaction_costs`)**:
   - Penalizes frequent trading by incorporating proportional costs for portfolio turnover.
   - Higher values discourage frequent rebalancing and reduce costs.
2. **Diversification Penalty (`delta` and `lambda`)**:
   - Encourages broader distribution of portfolio weights, reducing over-concentration in a few assets.
   - Higher values lead to more evenly spread weights.
   - Target value set at `delta`.
3. **Leverage Constraint (`leverage` and `eta`)**:
   - Encourages balanced exposure by limiting total portfolio leverage.
   - Higher values of `eta` restrict leverage, reducing risk and instability.
   - Target value set at `leverage`.

Optimization Goals

By adjusting these parameters, `sharpe_ratio_loss` balances:

- Maximizing the Sharpe Ratio.
- Adhering to practical constraints like turnover limits, diversification, and leverage caps.

Transaction Cost Penalty

Frequent portfolio rebalancing incurs costs that can erode returns. By setting `transaction_costs`, the loss function penalizes high turnover.

```
# Adjust transaction cost penalty in base configuration
config_transaction_cost <- config_box_sigmoid
config_transaction_cost$loss$transaction_costs <- 0.005  # Proportional cost for each transaction

# Backtest
portfolios_transaction_cost <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  features = features,
  pf_config = list(keras_weights_transaction_cost = list(weight_func = "keras_weights", config1 =
         config_transaction_cost)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
```

```
  verbose = verbose
)
```

Diversification Penalty

Over-concentration in a few assets increases risk. By applying the `lambda` parameter, the loss function encourages more evenly distributed weights.

```
# Adjust diversification penalty in base configuration
config_diversification <- config_box_sigmoid
config_diversification$loss$lambda <- 0.5  # Encourage even allocation
config_diversification$loss$delta <- 0.0289  # Target value for diversification

# Backtest
portfolios_diversification <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  features = features,
  pf_config = list(keras_weights_diversification = list(weight_func = "keras_weights", config1 =
        config_diversification)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Leverage Constraint

Excessive leverage amplifies risk and can lead to instability. The `leverage` parameter limits total portfolio exposure.

Configuration and Backtesting

```
# Adjust leverage constraint in base configuration
config_leverage <- config_box_sigmoid
config_leverage$loss$leverage <- 0.8  # Encourage leverage at 0.8 (not fully invested)
config_leverage$loss$eta <- 0.5  # Leverage penalty

# Backtest
portfolios_leverage <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  features = features,
  pf_config = list(keras_weights_leverage = list(weight_func = "keras_weights", config1 =
        config_leverage)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Combined Constraints

```
# Combine all penalties in base configuration
config_combined <- config_box_sigmoid
config_combined$loss$transaction_costs <- 0.005
config_combined$loss$lambda <- 0.5
config_combined$loss$delta <- 0.0289
config_combined$loss$leverage <- 1.0
config_combined$loss$eta <- 0.5

# Backtest
portfolios_combined <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  features = features,
  pf_config = list(keras_weights_combined = list(weight_func = "keras_weights", config1 =
        config_combined)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Comparative Analysis

```r
# Combine portfolios for comparison
combined_portfolios2 <- combine_portfolioReturns(list(
  portfolios_postprocessed,
  portfolios_box_sigmoid,
  postprocessing_portfolios(portfolios_box_sigmoid, pp_config),
  portfolios_transaction_cost,
  postprocessing_portfolios(portfolios_transaction_cost, pp_config),
  portfolios_diversification,
  postprocessing_portfolios(portfolios_diversification, pp_config),
  portfolios_leverage,
  postprocessing_portfolios(portfolios_leverage, pp_config),
  portfolios_combined,
  postprocessing_portfolios(portfolios_combined, pp_config)
))

# Summarize combined weights
tt <- summary.weights(postprocessing_portfolios(combined_portfolios2, pp_config), print = TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_simple_1	0/0.0102	0.004 (0.0038)	0.0036 (0.001)	0.0113 (0.0019)	91.8045	1.5 (0)	0.1059 (0.0095)	0 (0)	0.2892	55.5298
keras_weights_box_sigmoid_2	0/0.1	0.0024 (0.0071)	0.0068 (0.0021)	0.0209 (0.0102)	65.1258	0.8755 (0.1718)	0.1396 (0.0371)	0 (0)	0.2426	46.5831
keras_weights_box_sigmoid_3	0/0.1	0.0024 (0.0072)	0.0069 (0.0022)	0.0216 (0.011)	64.3246	0.8898 (0.1606)	0.1417 (0.0392)	0 (0)	0.2453	47.1048
keras_weights_transaction_cost_4	0/0.1	0.0031 (0.0062)	0.0059 (0.0017)	0.0178 (0.0067)	66.1526	1.1432 (0.3505)	0.1311 (0.0252)	0 (0)	0.2362	45.3445
keras_weights_transaction_cost_5	0/0.1	0.0031 (0.0062)	0.0059 (0.0018)	0.0181 (0.0072)	65.9436	1.1469 (0.3464)	0.1318 (0.0265)	0 (0)	0.2367	45.4369
keras_weights_diversification_6	0/0.1	0.0024 (0.0067)	0.0065 (0.0017)	0.019 (0.0082)	66.0064	0.906 (0.1797)	0.1345 (0.0309)	0 (0)	0.2428	46.6130
keras_weights_diversification_7	0/0.1	0.0024 (0.0068)	0.0065 (0.0018)	0.0193 (0.0085)	65.7359	0.9108 (0.176)	0.1352 (0.0317)	0 (0)	0.2435	46.7467
keras_weights_leverage_8	0/0.1	0.0024 (0.0071)	0.0068 (0.0021)	0.0209 (0.0102)	65.1258	0.8755 (0.1718)	0.1396 (0.0371)	0 (0)	0.2426	46.5831
keras_weights_leverage_9	0/0.1	0.0024 (0.0072)	0.0069 (0.0022)	0.0216 (0.011)	64.3246	0.8898 (0.1606)	0.1417 (0.0392)	0 (0)	0.2453	47.1048
keras_weights_combined0_10	0/0.1	0.0032 (0.006)	0.0057 (0.0016)	0.0174 (0.0065)	67.1589	1.1997 (0.3231)	0.1295 (0.0245)	0 (0)	0.2444	46.9214
keras_weights_combined1_11	0/0.1	0.0032 (0.006)	0.0058 (0.0016)	0.0174 (0.0066)	67.1112	1.2004 (0.3221)	0.1297 (0.0247)	0 (0)	0.2444	46.9280

Observation: The weight summary shows:

- Most constrained portfolios show similar levels of diversification and turnover.
- It is hard from this analysis to see the impact of the leverage constraint, as the leverage is already at 1.0 for the unconstrained portfolios.

```r
# Summarize combined performance
tt <- summary.performance2(postprocessing_portfolios(combined_portfolios2, pp_config), test = TRUE,
         print = TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_simple_1	0.291**	0.8973**	0.6848	55.5298	0.8868**	0.0818***	0.1332
keras_weights_box_sigmoid_2	0.2459**	1.2137***	0.6188	46.5831	1.2***	0.0726***	0.1843
keras_weights_box_sigmoid_3	0.2534**	1.2246***	0.6239	47.1048	1.211***	0.0778***	0.1891
keras_weights_transaction_cost_4	0.2844***	1.2724***	0.5945	45.3445	1.2601***	0.1022***	0.2095
keras_weights_transaction_cost_5	0.2857***	1.2767***	0.5958	45.4369	1.2644***	0.1031***	0.2106
keras_weights_diversification_6	0.2525**	1.2325***	0.6083	46.6130	1.219***	0.0823***	0.1895
keras_weights_diversification_7	0.2551**	1.2394***	0.6099	46.7467	1.226***	0.0844***	0.1915
keras_weights_leverage_8	0.2459**	1.2137***	0.6188	46.5831	1.2***	0.0726***	0.1843
keras_weights_leverage_9	0.2534**	1.2246***	0.6239	47.1048	1.211***	0.0778***	0.1891
keras_weights_combined0_10	0.3036***	1.2493***	0.5946	46.9214	1.2376***	0.1074***	0.2150
keras_weights_combined1_11	0.3039***	1.2504***	0.5948	46.9280	1.2388***	0.1077***	0.2153
benchmark	0.1533	0.7955	0.0000	16.4426	0.7902	NaN	0.0976

Observation: The performance summary reveals:

- All constrained portfolios exhibit Sharpe ratios (out-of-sample and annualized) significantly above 1, even when taking transaction costs properly into account. All cases therefore outperform the equally weighted portfolio.
- The portfolios with the highest turnover-adjusted Sharpe ratios are the combined constraints and the transaction cost constrained portfolios, with the highest Sharpe ratios being shown for the latter portfolios.
- Postprocessing has no impact on the combined portfolio, as all features of this portfolio already perfectly adhere to the constraints.

Conclusion

In the entire exercise we have learned how to use keras to predict and backtest portfolio weights, that - even taking into account data leakage and avoiding too frequent reestimation - significantly outperform the equally weighted benchmnark. We have also seen how to enforce practical constraints on the portfolio weights, leading to more realistic allocations and better performance. The most significant improvements were achieved by penalizing the loss function with transaction costs, leading to the overall highest Sharpe ratio (by a samll margin).

In the following we will have a look at benchmark portfolios.

Benchmark-Based Portfolio Optimization

Now we come to the even more difficult case of tilting a benchmark in order to outperform it. Many portfolio strategies aim to outperform a benchmark, the **InvestigatoR** package provides tools to achieve this:

1. Loss functions like `sharpe_ratio_difference_loss` and `information_ratio_loss_active_returns`.
2. Customizable activation functions to enforce constraints, including weight changes relative to the benchmark.

This section demonstrates:

1. Starting with an unconstrained benchmark case.
2. Introducing activation functions for benchmark-relative weights.
3. Analyzing the impact of L1 and L2 penalties.

We begin by setting up a model, which will be used to optimize the portfolio tilts (delta_weights). This model will not enforce any constraints, leading to unrealistic allocations (not entirely, as the used output activation will enforce weights between -1 and 1). More specifically, we define a set of portfolio constraints for backtesting:

- **Long-Short**: Explicit short-selling; tilting portfolio must be self-financing.
- **Weight Constraints**: Individual weights must be smaller than 10% in absolute value.
- **Leverage Limit**: Total exposure is capped at 1.0 on both sides (negative weights and positive weights).

Basic setup (based on increasing Sharpe ratio vis-a-vis the benchmark)

Benchmark Definition and Data Setup

We begin by adding a value-weighted benchmark to the dataset `data_ml_red`.

```
# Add benchmark to dataset
data_ml_red <- data_ml_red %>%
  group_by(date) %>%
  mutate(benchmark = Mkt_Cap_3M_Usd / sum(Mkt_Cap_3M_Usd)) %>%
  ungroup()
```

The benchmark represents a dynamic, market-cap-weighted portfolio used as a reference for optimization.

Unconstrained Benchmark Optimization

Objective

Optimize portfolio weights to maximize the Sharpe Ratio difference from the benchmark without imposing constraints on weight changes.

Configuration and Backtesting

```
# Define a simple Keras model configuration
# here with different activation function, because we also need negative weights
config_unconstrained_benchmark <- list(
  layers = list(
    list(type = "dense", units = 32, activation = "linear"),
    list(type = "dense", units = 16, activation = "tanh"),
    list(type = "dense", units = 1, activation = "tanh")  # No custom activation
  ),
  loss = list(
    name = "sharpe_ratio_difference_loss", # Standard Sharpe Ratio Difference loss with no penalties
    lambda_l1 = 0,  # No L1 or L2 penalties
    lambda_l2 = 0
  ),
  optimizer = list(name = "optimizer_adam", learning_rate = 0.001),
  epochs = 50,
```

```
  verbose = 0,
  seeds = c(42)  # Set random seed for reproducibility
)
```

```
# Backtest
portfolios_bm_unconstrained <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  benchmark_label = "benchmark",
  features = features,
  pf_config = list(keras_weights_unconstrained = list(weight_func = "keras_weights", config1 =
         config_unconstrained_benchmark)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Results

```
# Define postprocessing configuration
pp_config_bm <- list(list(operation = "set_weightsum", sum = 0, min_weight = -0.1, max_weight = 0.1,
         min_sum = -1, max_sum = 1, allow_short_sale = TRUE))
# combine unconstrained and constrained portfolios
combined_portfolios_bm <- combine_portfolioReturns(list(
  portfolios_bm_unconstrained,
  postprocessing_portfolios(portfolios_bm_unconstrained, pp_config_bm))
  )

# Summarize weights
tt <- summary.weights(combined_portfolios_bm, print = TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_unconstrained_1	-0.8489/0.9605	0.0474 (0.3514)	0.3462 (0.0243)	46.767 (6.1439)	0.0217	109.8732 (8.3576)	6.8246 (0.4387)	178.1927 (34.0683)	23.6827	4547.077
keras_weights_unconstrained_2	-0.0082/0.0139	0.0027 (0.0048)	0.0048 (2e-04)	0.0113 (9e-04)	89.0878	1.7507 (0.1299)	0.1062 (0.0042)	151.9792 (18.6113)	0.3917	75.2037

Observation: The weight summary shows:

- Large weight differences -0.8489/0.9605 (and -0.0082/0.0139 after postprocessing), indicating extreme allocations that are smoothed after postprocessing (note, that postprocessing is necessary here, as the optimizer cannot guarantee, that benchmark tilts are self-financing)
- Large total turnover at 4547.0773, indicating excessive trading activity relative to the benchmark at 75.2037
- Concentration in a few assets, leading to poor diversification, based on the average Herfindahl-Hirschman Index (HHI) of 46.767 (6.1439) (and 0.0113 (9e-04) after postprocessing).

```
tt <- summary.performance2(combined_portfolios_bm, transaction_cost = 0.005, gamma = 3, test = TRUE,
         print = TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_unconstrained_1	10.5208***	1.38***	55.0888	4547.0773	1.1941***	0.1152***	-76.6569
keras_weights_unconstrained_2	0.1981**	0.864***	1.0000	75.2037	0.7629**	0.0855***	0.1193
benchmark	0.0938	0.5133	0.0000	17.1261	0.4839	NaN	0.0437

Observation: Performance-wise we observe the following:

- The Sharpe ratio is high at 1.38*** (0.864*** for the postprocessed portfolio), larger than the value weighted (given) benchmark at 0.5133, therefore significantly exceeding it for the raw as well as for the postprocessed portfolio.
- The turnover, reduces down to 75 after postprocessing, indicating excessive trading activity relative to the benchmark at 17. This leads to a turnover-adjusted Sharpe Ratio of 1.1941*** (0.7629** for the postprocessed portfolio), which is still significantly above the benchmark at 0.4839. Also, and most importantly, the information ratio is significant at 0.1152*** (0.0855*** for the postprocessed portfolio), indicating that the portfolio is able to outperform the benchmark significantly in raw and postprocessed case, even when taking into account tracking error.

Constrained Benchmark Optimization

In the next case, we restrict the portfolio weights to ensure they align with the benchmark. We apply activation functions to enforce constraints on the benchmark-relative weights.

Apply constraints:

- Weights bounded between -0.1 and 0.1.

Updated Activation Configuration

```
# Define a new activation for constrained benchmark-relative weights
config_constrained_bm_activation <- config_unconstrained_benchmark
config_constrained_bm_activation$layers[[3]]$activation <- activation_box_sigmoid(
  min_weight = -0.1,
  max_weight = 0.1
)
```

Backtesting

```
# Backtest with constrained activation
portfolios_bm_constrained <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  benchmark_label = "benchmark",
  features = features,
  pf_config = list(keras_weights_constrained = list(weight_func = "keras_weights", config1 =
         config_constrained_bm_activation)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Results

```
# combine unconstrained and constrained portfolios
combined_portfolios_bm2 <- combine_portfolioReturns(list(
  postprocessing_portfolios(portfolios_bm_unconstrained, pp_config_bm),
  portfolios_bm_constrained,
  postprocessing_portfolios(portfolios_bm_constrained, pp_config_bm)
  )
  )

# Summarize weights and performance
tt <- summary.weights(combined_portfolios_bm2, print = TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_unconstrained_1	-0.0082/0.0139	0.0027 (0.0048)	0.0048 (2e-04)	0.0113 (9e-04)	89.0878	1.7507 (0.1299)	0.1062 (0.0042)	151.9792 (18.6113)	0.3917	75.2037
keras_weights_constrained_2	-0.0567/0.0764	0.0038 (0.0217)	0.0213 (0.002)	0.1807 (0.0375)	5.8220	6.5745 (0.853)	0.4226 (0.0457)	179.3073 (34.6778)	1.4420	276.8555
keras_weights_constrained_3	-0.0132/0.0254	0.0027 (0.0056)	0.0056 (4e-04)	0.0142 (0.0019)	71.3310	1.7612 (0.0698)	0.1191 (0.0077)	132.3958 (14.5055)	0.4121	79.1196

Observation: The weight summary shows:

- The constrained portfolios exhibit more stable weight distributions, with average standard deviations of 0.0213 (0.002) and 0.0056 (4e-04) after postprocessing.
- The average Herfindahl-Hirschman Index (HHI) is now 0.1807 (0.0375) and 0.0142 (0.0019) after postprocessing, indicating worse diversification, than for the postprocessed unconstrained portfolio.
- The total turnover is 276.8555 and 79.1196 after postprocessing, indicating a more stable portfolio.

```
tt <- summary.performance2(combined_portfolios_bm2, transaction_cost = 0.005, gamma = 3, test = TRUE,
         print = TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_unconstrained_1	0.1981**	0.864***	1.000	75.2037	0.7629**	0.0855***	0.1193
keras_weights_constrained_2	0.7437***	1.4499***	3.468	276.8555	1.2832***	0.1194***	0.3490
keras_weights_constrained_3	0.2687***	1.0595***	1.000	79.1196	0.9636***	0.1033***	0.1722
benchmark	0.0938	0.5133	0.000	17.1261	0.4839	NaN	0.0437

Observation: Performance-wise we observe the following:

- The Sharpe ratio is highest at 1.4499*** (1.0595*** for the postprocessed portfolio), significantly exceeding the benchmark at 0.5133, and larger than the unconstrained and postprocessed portfolio.

- Even when taking transaction costs into account, the turnover-adjusted Sharpe Ratio is significant at 1.2832*** (0.9636*** for the postprocessed portfolio), which is still significantly above the benchmark at 0.4839.
- The information ratio is significant at 0.1194*** (0.1033*** for the postprocessed portfolio), indicating that the portfolio is able to outperform the benchmark significantly in raw and postprocessed case, even when taking into account tracking error.

```
tt <- summary.performance(combined_portfolios_bm2, test = TRUE, print = TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Annualized Volatility	Sharpe Ratio	Sortino Ratio	Max Drawdown	Skewness	Kurtosis	Value at Risk	Conditional Value at Risk	Alpha
keras_weights_unconstrained_1	0.1981***	0.2292	0.864***	0.4012	0.5936	-0.2331	5.5958	-0.0889	-0.1660	0.0077***
keras_weights_constrained_2	0.7437***	0.5130	1.4499***	0.8461	0.7508	0.2392	2.9473	-0.1620	-0.2416	0.0514***
keras_weights_constrained_3	0.2687***	0.2536	1.0595***	0.5220	0.6005	-0.1290	5.1067	-0.0928	-0.1668	0.0133***
benchmark	0.0938**	0.1827	0.5133	0.2192	0.5281	-0.4639	2.5386	-0.0828	-0.1373	0

[1] “Portfolio” “Annualized Mean” “Annualized Volatility” “Sharpe Ratio” “Sortino Ratio” “Max Drawdown”
[7] “Skewness” “Kurtosis” “Value at Risk” “Conditional Value at Risk” “Alpha” “Beta”
[13] “Tracking Error” “Information Ratio”

Observation: Looking at further details, performance-wise we observe:

- Here we find the reason for the larger information ratio of the postprocessed portfolio in relation to the regular constrained portfolio: The alpha is smaller at 0.0514*** to 0.0133***, while the tracking error is also smaller at 0.4536 to 0.1411. This leads to a higher information ratio of 1.6404*** to 1.3064***.
- The beta is significantly higher for the constrained portfolios at 1.3585*** to 1.1651***, indicating that the portfolios are more prone to systematic or benchmark risk.

Penalties for `sharpe_ratio_difference_loss`

The `sharpe_ratio_difference_loss` function supports the application of:

- **L1 Penalty (`lambda_l1`)**: Encourages sparse deviations from the benchmark, reducing the number of assets with non-zero weight changes.
- **L2 Penalty (`lambda_l2`)**: Promotes smoother weight deviations, discouraging extreme differences.

In this step, we:

1. Apply L1 and L2 penalties separately.
2. Combine L1 and L2 penalties.
3. Analyze the results to understand the impact of each penalty.

L1 penalty

```
# Apply L1 penalty in the configuration
config_l1 <- config_constrained_bm_activation
config_l1$loss$lambda_l1 <- 0.1  # Apply L1 penalty
```

```
# Backtest with L1 penalty only
portfolios_l1 <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  benchmark_label = "benchmark",
  features = features,
  pf_config = list(keras_weights_l1 = list(weight_func = "keras_weights", config1 = config_l1)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

L2 penalty

```
# Apply L2 penalty in the configuration
config_l2 <- config_constrained_bm_activation
config_l2$loss$lambda_l2 <- 0.1   # Apply L2 penalty
```

```
# Backtest with L2 penalty only
portfolios_l2 <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  benchmark_label = "benchmark",
  features = features,
  pf_config = list(keras_weights_l2 = list(weight_func = "keras_weights", config1 = config_l2)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
```

```
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Combined L1 and L2 Penalties

```
# Apply both L1 and L2 penalties
config_l1_l2 <- config_constrained_bm_activation
config_l1_l2$loss$lambda_l1 <- 0.1  # Apply L1 penalty
config_l1_l2$loss$lambda_l2 <- 0.1   # Apply L2 penalty
```

```
# Backtest with combined L1 and L2 penalties
portfolios_l1_l2 <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  benchmark_label = "benchmark",
  features = features,
  pf_config = list(keras_weights_l1_l2 = list(weight_func = "keras_weights", config1 = config_l1_l2)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Comparative Analysis

```
# Combine portfolios for comparison
combined_penalty_portfolios <- combine_portfolioReturns(list(
  portfolios_bm_constrained,
  portfolios_l1,
  portfolios_l2,
  portfolios_l1_l2)
)

# Summarize combined weights
# be aware that we need to postprocess to get zero-sum tilting portfolios
tt <- summary.weights(postprocessing_portfolios(combined_penalty_portfolios, pp_config_bm), print =
         TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_constrained_1	-0.0132/0.0254	0.0027 (0.0056)	0.0056 (4e-04)	0.0142 (0.0019)	71.3310	1.7612 (0.0698)	0.1191 (0.0077)	132.3958 (14.5055)	0.4121	79.1196
keras_weights_l1_2	-0.0412/0.0497	0.0027 (0.0074)	0.0074 (3e-04)	0.0228 (0.0017)	44.0571	2.2131 (0.0533)	0.1509 (0.0054)	121.3438 (5.5375)	0.8610	165.3202
keras_weights_l2_3	-0.0312/0.0436	0.0027 (0.0066)	0.0066 (3e-04)	0.019 (0.0016)	53.0908	2.04 (0.051)	0.1376 (0.0059)	126.901 (5.9395)	0.6376	122.4146
keras_weights_l1_l2_4	-0.0409/0.0489	0.0027 (0.0073)	0.0073 (3e-04)	0.0228 (0.0016)	44.1674	2.2161 (0.0532)	0.1507 (0.0054)	121.1042 (5.5864)	0.8575	164.6331

Observation: The weight summary shows:

- The L1 penalty leads to more sparse weight distributions, with average standard deviations of 0.0074 (3e-04) and 0.0073 (3e-04).
- The L2 penalty results in smoother weight distributions, with average standard deviations of 0.0066 (3e-04) and 0.0073 (3e-04).
- The total turnover is lowest for the L1 penalty at 165.3202, followed by the combined L1 and L2 penalty at 164.6331 and the L2 portfolio at 122.4146.
- Diversification (in terms of diversity) is best for the L2 penalty at 53.0908, followed by the L1 penalty at 44.0571 and the the combined L1 and L2 portfolio.

```
# Summarize combined performance
tt <- summary.performance2(postprocessing_portfolios(combined_penalty_portfolios, pp_config_bm),
         transaction_cost = 0.005, gamma = 3, test = TRUE, print = TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_constrained_1	0.2687***	1.0595***	1	79.1196	0.9636***	0.1033***	0.1722
keras_weights_l1_2	0.0794	0.4132	1	165.3202	0.1444	-0.0079	0.0240
keras_weights_l2_3	0.1898**	0.8325**	1	122.4146	0.6657	0.0553***	0.1118

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_l1_l2_4	0.0765	0.3993	1	164.6331	0.1307	-0.0096	0.0215
benchmark	0.0938	0.5133	0	17.1261	0.4839	NaN	0.0437

Observation: Performance-wise we observe the following:

- The Sharpe ratio is highest for the constrained portfolio at 1.0595*, **significantly exceeding the benchmark at 0.5133 and larger than the portfolios with L1 and L2 penalties. The L1 and L2 penalties lead to Sharpe ratios of 0.4132 and 0.8325**, respectively.
- The turnover-adjusted Sharpe Ratio is highest for the constrained portfolio at 0.9636***, followed by the L1 and L1/L2 portfolios at 0.1444 and 0.1307.
- The information ratio is highest for the constrained portfolio at 0.1033, ***followed by the L2 and L1/L2 portfolios at 0.0553*** and -0.0096.

Information Ratio Loss Optimization

The `information_ratio_loss` function focuses on optimizing the Information Ratio:

- **Information Ratio (IR)**: The active return divided by the tracking error (volatility of active return relative to the benchmark).
- This loss function allows customization through penalties like L1 and L2, balancing active return and risk.

This section demonstrates:

1. **Non-regression-based** `information_ratio_loss`: Optimize IR with constraints.
2. **Regression-based** `information_ratio_loss`: Adds alpha and beta modeling for enhanced optimization.
3. Applying L1 and L2 penalties together to assess their impact.

Non-Regression-Based `information_ratio_loss`

Optimize the Information Ratio without regression constraints while applying L1 and L2 penalties to encourage sparse and smooth deviations.

```
# Define configuration for non-regression information_ratio_loss
config_ir_non_reg <- config_constrained_bm_activation
config_ir_non_reg$loss$name <- "information_ratio_loss_active_returns"
config_ir_non_reg$loss$lambda_l1 <- 0.1  # Apply L1 penalty
config_ir_non_reg$loss$lambda_l2 <- 0.1   # Apply L2 penalty
```

```
# Backtest with non-regression information_ratio_loss
portfolios_ir_non_reg <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  benchmark_label = "benchmark",
  features = features,
  pf_config = list(keras_weights_ir_non_reg = list(weight_func = "keras_weights", config1 =
         config_ir_non_reg)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
  verbose = verbose
)
```

Regression-Based `information_ratio_loss`

The regression-based version of `information_ratio_loss` introduces alpha and beta modeling:

- **Alpha**: The component of returns uncorrelated with the benchmark.
- **Beta**: Sensitivity of portfolio returns to benchmark returns.

By explicitly modeling alpha and beta, this version improves the interpretability and precision of IR optimization.

```
# Define configuration for regression-based information_ratio_loss
config_ir_reg <- config_ir_non_reg
config_ir_reg$loss$name <- "information_ratio_loss_regression_based"
```

```
# Backtest with regression-based information_ratio_loss
portfolios_ir_reg <- backtesting_weights(
  data = data_ml_red,
  return_label = return_label,
  benchmark_label = "benchmark",
  features = features,
  pf_config = list(keras_weights_ir_reg = list(weight_func = "keras_weights", config1 =
         config_ir_reg)),
  rolling = rolling,
  window_size = window_size,
  step_size = step_size,
  offset = offset,
  num_cores = num_cores,
```

```
  verbose = verbose
)
```

Combined Analysis

Comparing Non-Regression and Regression-Based Results

```
# Combine portfolios for comparison
combined_ir_portfolios <- combine_portfolioReturns(list(
  portfolios_bm_constrained,
  portfolios_l1_l2,
  portfolios_ir_non_reg,
  portfolios_ir_reg
))

# Summarize combined weights
tt <- summary.weights(postprocessing_portfolios(combined_ir_portfolios, pp_config_bm), print = TRUE)
```

Portfolio Weights Summary with Advanced Metrics

Portfolio	Weights (Min/Max)	Weights M(SD)	Std.Dev./Date M(SD)	HHI M(SD)	Diversity	L1 Norm M(SD)	L2 Norm M(SD)	Avg. Short Weights M(SD)	Average Turnover	Total Turnover
keras_weights_constrained_1	-0.0132/0.0254	0.0027 (0.0056)	0.0056 (4e-04)	0.0142 (0.0019)	71.3310	1.7612 (0.0698)	0.1191 (0.0077)	132.3958 (14.5055)	0.4121	79.1196
keras_weights_l1_l2_2	-0.0409/0.0489	0.0027 (0.0073)	0.0073 (3e-04)	0.0228 (0.0016)	44.1674	2.2161 (0.0532)	0.1507 (0.0054)	121.1042 (5.5864)	0.8575	164.6331
keras_weights_ir_non_reg_3	-0.0413/0.0483	0.0027 (0.0074)	0.0074 (3e-04)	0.0228 (0.0016)	44.0251	2.2228 (0.0536)	0.151 (0.0053)	120.8854 (5.6925)	0.8603	165.1858
keras_weights_ir_reg_4	-0.0411/0.048	0.0027 (0.0073)	0.0073 (3e-04)	0.0226 (0.0016)	44.4829	2.2146 (0.0547)	0.1502 (0.0054)	120.9688 (5.2954)	0.8540	163.9593

Observation: The weight summary shows:

- The constrained portfolios exhibit more stable weight distributions, with average standard deviations of 0.0074 (3e-04) and 0.0073 (3e-04).
- The average Herfindahl-Hirschman Index (HHI) is equally low for all portfolios, indicating better diversification.
- The total turnover is lowest for the L1 and L2 penalties at 164.6331, followed by the regression-based portfolio at 163.9593.
- All in all, the portfolios show not many distinguishing factors.

```
# Summarize combined performance
tt <- summary.performance2(postprocessing_portfolios(combined_ir_portfolios, pp_config_bm), test =
        TRUE, print = TRUE)
```

Portfolio Performance Summary

Portfolio	Annualized Mean	Sharpe Ratio	Active Share	Turnover	TC Adjusted Sharpe	Information Ratio	CER (gamma=3)
keras_weights_constrained_1	0.2687***	1.0595***	1	79.1196	1.0404***	0.1033***	0.1722
keras_weights_l1_l2_2	0.0765	0.3993	1	164.6331	0.3456	-0.0096	0.0215
keras_weights_ir_non_reg_3	0.0724	0.3786	1	165.1858	0.3246	-0.0119	0.0176
keras_weights_ir_reg_4	0.0799	0.421	1	163.9593	0.367	-0.0077	0.0258
benchmark	0.0938	0.5133	0	17.1261	0.5074	NaN	0.0437

Observation: Performance-wise we observe the following:

- The constrained portfolio exhibits the highest Sharpe ratio at 1.0595***, significantly exceeding the benchmark at 0.5133 and larger than the portfolios with information ratio optimization.
- The turnover-adjusted Sharpe Ratio is highest for the constrained portfolio at 1.0404***, followed by the regression-based portfolio at 0.367.
- The information ratio is highest for the constrained portfolio at 0.1033***, followed by the non-regression and regression-based portfolios at -0.0119 and -0.0077.
- The Certainty Equivalent Return (CER) is highest for the constrained portfolio at , followed by the non-regression and regression-based portfolios at and .
- Most interestingly, its not the information ratio based optimal portfolios that shows the highest information rate.

Conclusion

In this notebook, we have demonstrated how to optimize portfolio weights using Keras models in R, focusing on practical constraints and benchmark-relative optimization. We have covered various portfolio optimization types plus different loss functions.

Appendix

Detailed Metrics, Activation Functions, and Postprocessing Routines

Available Metrics

1. **Portfolio Metrics (via `summary.performance`)**
   - **Absolute Metrics:**
     - Annualized Mean Return
     - Annualized Volatility
     - Sharpe Ratio (with Memmel test)
     - Sortino Ratio
     - Maximum Drawdown
     - Skewness and Kurtosis
     - Value at Risk (VaR) and Conditional VaR (CVaR)
   - **Relative Metrics:**
     - Alpha, Beta (CAPM)
     - Tracking Error
     - Information Ratio (IR with p-values)
   - **Additional Metrics:**
     - Transaction Cost Adjusted Sharpe Ratio
     - Certainty Equivalent Return (adjusted for risk aversion, γ)
2. **Weight-Based Metrics (via `summary.weights`)**
   - **Statistical Summaries:**
     - Weight Mean and Standard Deviation (per date and portfolio)
     - Minimum and Maximum Weight
   - **Diversity Measures:**
     - Herfindahl-Hirschman Index (HHI)
     - Diversity Inverse of HHI
   - **Norms:**
     - L1 Norm (sum of absolute weights)
     - L2 Norm (Euclidean norm)
   - **Short Positions:**
     - Count of Short Weights (negative weights)
   - **Turnover Analysis:**
     - Average Turnover Per Period
     - Total Turnover
3. **Correlation Analysis (via `summary.correlation`)**
   - Correlation matrix between portfolio returns and a benchmark or among portfolios.

Activation Functions and Constraints

1. **Portfolio Weight Normalization (Set Weights Sum):**
   - Ensures weights sum to a specific value (e.g., 1 for fully invested portfolios).
   - Handles short-sale constraints, min/max weights, and bounds on sum.
2. **Weight Regularization (Flatten Weights):**
   - **L1 Regularization:** Shrinks weights towards zero (lasso-like behavior).
   - **L2 Regularization:** Penalizes large weights (ridge-like behavior).
   - **Elastic Net Mixing:** Combines L1 and L2 regularization using a mixing parameter.
3. **Turnover Reduction:**
   - **Linear Smoothing:** Incrementally adjusts weights toward previous values.
   - **Exponential Smoothing:** Weighted adjustment favoring recent weights.
4. **Diversification Enhancement:**
   - Targets an HHI threshold for diversity.
   - Adjusts weights to meet desired concentration limits.

Postprocessing Routines

1. **Weight Adjustments:**
   - Apply constraints (e.g., normalization, bounds).
   - Adjust for regularization (L1/L2) and diversity (HHI targets).
   - Reduce portfolio turnover to minimize transaction costs.
2. **Recalculation of Portfolio Returns:**
   - Compute new portfolio returns after weight adjustments.
   - Integrate with benchmark returns if applicable.

Summary Functions

1. `summary.performance`**:**
   - Provides a comprehensive analysis of portfolio metrics.
   - Includes statistical tests for Sharpe ratio, alpha, and IR significance.
   - Outputs results in a structured table with optional significance stars.
2. `summary.weights`**:**
   - Computes detailed weight statistics and turnover metrics.

- Summarizes portfolio diversity and weight constraints.

3. `summary.correlation`:
   - Calculates correlation among portfolio returns or against a benchmark.
   - Produces an easy-to-read correlation matrix.
4. `summary.performance2`:
   - Advanced portfolio performance with transaction costs and certainty equivalent returns.
5. `combine_portfolioReturns`:
   - Combines multiple `portfolioReturns` objects into one, ensuring consistency across benchmarks and models.

Visualization (`plot.portfolioReturns`)

- Plots cumulative returns for multiple portfolios.
- Supports additional PerformanceAnalytics visualizations (e.g., wealth index, rolling statistics).

Certainly! Let's delve deeper into the various metrics used in portfolio analysis, particularly those that assess distance from a benchmark, regularization, and other key performance factors.

Keras Metrics

1. Distance from Benchmark Metrics

L1 Distance from Benchmark (`distance_from_benchmark_l1_metric`)

- **Definition**: The sum of the absolute differences between portfolio weights and benchmark weights.
- **Formula**:

$$\text{L1 Distance} = \sum_{i=1}^{n} \left| w_i^{\text{portfolio}} - w_i^{\text{benchmark}} \right|$$

- **Use Case**:
  - Measures deviation from the benchmark in a straightforward way.
  - Helps control active risk by limiting how far portfolio weights deviate from the benchmark.
  - Useful for minimizing tracking error in a constrained optimization setting.

L2 Distance from Benchmark (`distance_from_benchmark_l2_metric`)

- **Definition**: The Euclidean distance between portfolio weights and benchmark weights.
- **Formula**:

$$\text{L2 Distance} = \sqrt{\sum_{i=1}^{n} \left(w_i^{\text{portfolio}} - w_i^{\text{benchmark}} \right)^2}$$

- **Use Case**:
  - Penalizes larger deviations more heavily than L1.
  - Smooths weight differences, useful for optimizing risk-adjusted performance.

Tracking Error (TE)

- **Definition**: The standard deviation of the difference between portfolio returns and benchmark returns over time.
- **Formula**:

$$\text{TE} = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \left(r_t^{\text{portfolio}} - r_t^{\text{benchmark}} \right)^2}$$

- **Use Case**:
  - Measures how closely the portfolio tracks the benchmark.
  - A low TE indicates a passive strategy, while a high TE suggests active management.

2. Regularization Metrics

L1 Norm of Portfolio Weights (`l1_weight_metric`)

- **Definition**: The sum of the absolute values of portfolio weights.
- **Formula**:

$$\text{L1 Norm} = \sum_{i=1}^{n} |w_i^{\text{portfolio}}|$$

- **Use Case**:
  - Encourages sparsity in portfolio weights, favoring fewer active positions.
  - Useful for managing transaction costs by limiting the number of trades.

L2 Norm of Portfolio Weights (`l2_weight_metric`)

- **Definition**: The square root of the sum of squared portfolio weights.
- **Formula**:

$$\text{L2 Norm} = \sqrt{\sum_{i=1}^{n} \left(w_i^{\text{portfolio}}\right)^2}$$

- **Use Case**:
  - Penalizes extreme positions and promotes weight regularization.
  - Helps prevent overconcentration in specific assets.

3. Performance Metrics

Sharpe Ratio

- **Definition**: Measures risk-adjusted return using the ratio of excess return to standard deviation.
- **Formula**:

$$\text{Sharpe Ratio} = \frac{\mathbb{E}[r^{\text{portfolio}} - r^{\text{risk-free}}]}{\sigma^{\text{portfolio}}}$$

- **Use Case**:
  - Evaluates overall portfolio performance relative to risk taken.

Sortino Ratio

- **Definition**: Similar to the Sharpe Ratio but only considers downside risk.
- **Formula**:

$$\text{Sortino Ratio} = \frac{\mathbb{E}[r^{\text{portfolio}} - r^{\text{risk-free}}]}{\sigma^{\text{downside}}}$$

- **Use Case**:
  - Better for asymmetrical return distributions or portfolios with minimal upside volatility.

Information Ratio (IR)

- **Definition**: Measures risk-adjusted active return relative to a benchmark.
- **Formula**:

$$\text{IR} = \frac{\mathbb{E}[r^{\text{portfolio}} - r^{\text{benchmark}}]}{\text{Tracking Error}}$$

- **Use Case**:
  - Assesses how much excess return is generated per unit of active risk.

4. Diversification Metrics

Herfindahl-Hirschman Index (HHI)

- **Definition**: Measures portfolio concentration.
- **Formula**:

$$\text{HHI} = \sum_{i=1}^{n} \left(w_i^{\text{portfolio}}\right)^2$$

- **Use Case**:
  - Low HHI indicates a well-diversified portfolio, while high HHI suggests concentration risk.

Active Share

- **Definition**: The proportion of portfolio weights differing from the benchmark.
- **Formula**:

$$\text{Active Share} = \frac{1}{2} \sum_{i=1}^{n} \left| w_i^{\text{portfolio}} - w_i^{\text{benchmark}} \right|$$

- **Use Case**:
  - A measure of active management; values closer to 1 indicate high divergence from the benchmark.

5. Risk Metrics

Maximum Drawdown

- **Definition**: The maximum observed loss from a portfolio's peak value.

- **Formula**:

$$\text{Max Drawdown} = \max_t \left(\frac{\text{Portfolio Value}_{\text{peak}} - \text{Portfolio Value}_t}{\text{Portfolio Value}_{\text{peak}}} \right)$$

- **Use Case**:
  - Measures worst-case risk exposure.

Value at Risk (VaR)

- **Definition**: The maximum expected loss at a certain confidence level.
- **Formula** (for a normal distribution):

$$\text{VaR} = -\mu + z_\alpha \cdot \sigma$$

  where z_α is the critical value for confidence level α.
- **Use Case**:
  - Quantifies tail risk.

Conditional VaR (CVaR)

- **Definition**: The expected loss beyond the VaR threshold.
- **Use Case**:
  - Addresses limitations of VaR by considering extreme losses.

6. Turnover Metrics

Turnover

- **Definition**: Measures the trading activity in a portfolio.
- **Formula**:

$$\text{Turnover} = \sum_{i=1}^{n} |w_i^{\text{current}} - w_i^{\text{previous}}|$$

- **Use Case**:
  - Tracks transaction costs and trading frequency.

Applications of Metrics

1. **Portfolio Optimization**:
   - Use L1 and L2 distances to control deviations from benchmarks.
   - Employ diversity metrics to avoid concentration risk.
   - Minimize turnover for cost-effective portfolio management.
2. **Backtesting**:
   - Evaluate historical performance using Sharpe, Sortino, and Information Ratios.
   - Analyze drawdowns, VaR, and CVaR for risk assessment.
3. **Regularization**:
   - Apply L1/L2 norms to constrain weights, balancing sparsity and concentration.
4. **Active Management**:
   - Use metrics like Active Share and IR to assess the effectiveness of active strategies.