

Exercise 4

In this exercise, the main goal was to create a machine learning algorithm using CTF characteristics from JKP and use this algorithm to create a long-short portfolio. The main evaluation mechanism is the Sharpe Ratio, which is used to compare how well the portfolio performs compared to other ML-based portfolios in the competition by Hellum et al. (2025).

The main idea is to predict next-month excess returns using firm characteristics and to construct a cross-sectional long-short portfolio based on predicted returns. I combine two common machine learning frameworks to create an ensemble approach using Linear Regularized Regression (also known as Ridge) and Random Forest. The main motivation behind choosing these two models is to capture both linear structures (using Ridge) and nonlinear interactions (using Random Forest).

Data

The dataset consists of three components including 1) `ctff_chars` (firm-level characteristics and next-month excess returns), 2) `ctff_features` (list of allowed features), and 3) `ctff_daily_ret` (daily excess returns). The number of observations are in the millions approximately equal to 103 million observations with roughly 400 characteristics. To ensure temporal correctness, I use next-month excess returns. Also, only variables included in the provided dataset were used, aligning with CTF constraints.

Feature Selection

Feature selection is another important consideration in ML-based algorithms since it can affect model accuracy. In this case, features were selected to ensure correctness; all unnecessary variables were excluded, such as identifiers, labels, the target variable, same-month return, and non-numeric columns. This ensures that manual feature selection based on prior knowledge of factor performance is restricted and well-known anomalies (e.g., momentum or value) are not given undue advantage.

Standardization

Standardization is used to preserve cross-sectional ordering and reduce the influence of extreme values in this large dataset. It also ensures comparability across time and avoids distributional assumptions. Moreover, in this case, standardization was performed month-by-month, preventing look-ahead bias, and was one of the most computationally heavy steps.

To ensure comparability across firms within each month, features were standardized cross-sectionally. First, missing values were imputed using the cross-sectional median for each month. Next, features were transformed using cross-sectional ranking by mapping all values into intervals of -1 and 1:

$$X_{i,t}^* = 2 \times \frac{\text{rank}(X_{i,t}) - 1}{N_t - 1} - 1$$

Model 1: Ridge Regression

Ridge regression was used as the primary linear model. Ridge is particularly appropriate here because the number of predictors is large, many characteristics are correlated, and it reduces overfitting relative to OLS.

The regularization parameter λ was selected using a time-consistent validation procedure by using a calibration window prior to the test period. Likewise, the final 24 months of that calibration sample were treated as validation, and the mean squared error was minimized over a grid of λ values. The selected λ was then held fixed throughout the test period to avoid overfitting. Training was performed using a rolling window that strictly included only data available before each prediction month and made computations far more efficient.

Model 2: Random Forest

To capture nonlinear relationships and higher-order interactions, a Random Forest model was implemented using the ranger package in RStudio. Random Forest builds an ensemble of decision trees where each tree is trained on a subsample of the data. Then, a random subset of predictors is considered at each split, and the final predictions are averages across trees.

In this context, the main advantage of using a Random Forest model is to capture nonlinear trends. It also automatically models interactions, is robust to multicollinearity, and does not require feature scaling (though standardization was retained for consistency). To control computational complexity and prevent overfitting, the model was retrained once every 12 months, the training sample was capped at a fixed maximum size, and a rolling window of past months was used.

Ensemble Strategy

The final prediction was constructed as a convex combination of ridge and random forest predictions using the following:

$$\hat{r}_{i,t} = w\hat{r}_{i,t}^{ridge} + (1 - w)\hat{r}_{i,t}^{RF}$$

with $w = 0.7$.

The main motivation to use an ensemble method was to benefit from both the models, where Ridge can capture linear factor structure and Random Forest captures nonlinearities. Moreover, averaging also reduces model variance, and ensemble methods are known to improve predictive stability in the literature due to their complexity.

Portfolio Construction

Finally, a long–short portfolio was constructed by first sorting firms by predicted return. Next, the top 20% were assigned equal positive weights, the bottom 20% were assigned equal negative weights, and the remaining firms received zero weight. The portfolio is dollar-neutral and rebalanced monthly. Also, no leverage or turnover constraints were imposed, consistent with the CTF rules. The annualized Sharpe ratio was calculated to evaluate portfolio performance.

Conclusion

The Sharpe Ratio is **0.8652**, which is comparable to the other dynamic ML models on the leadership board. Moreover, the model follows CTF rules by not using any future information, selecting features algorithmically, and avoiding look-ahead bias.

The results suggest that combining linear and nonlinear methods improves predictive performance relative to linear modeling alone. The ridge component captures broad factor exposures, while the random forest component identifies nonlinear exposures. The ensemble approach reduces estimation variance and improves stability across market regimes.

This study demonstrates that regularized linear models provide strong baseline predictive power and nonlinear machine learning models capture additional cross-sectional structure. The framework remains computationally feasible, fully rule-compliant, and economically interpretable.

APPENDIX

Link to OneDrive with Rcode and data files:

https://hankenfi-my.sharepoint.com/:f:/g/personal/minmuaha_hanken_fi/IgD9CzrmXhKvQa1lGJ5h_6KkAY-4Z_fXq3YFtlqcDMBEfE8?e=HtcfH3

Exercise 4 Code

```
##### Exercise 4 Code by Hassan Mir #####

#installing packages(c("DBI", "RPostgres", "keyring"))

keyring::key_set(service = "wrds", username = "hmir")

#running libraries

library(keyring)

library(DBI)

library(RPostgres)

#saving credentials

if (nrow(key_list("wrds"))==0) {

  stop(

    "No WRDS credentials stored.\n",

    "Run: keyring::key_set(service = 'wrds', username = 'WRDS_USERNAME')."

  )

}

wrds_un <- key_list(service = "wrds")$username[1]

wrds_pw <- key_get("wrds", wrds_un)

#connecting to WRDS

con <- dbConnect(

  RPostgres::Postgres(),
```

```

host   = "wrds-pgdata.wharton.upenn.edu",

port   = 9737,

dbname = "wrds",

sslmode = "require",

user    = wrds_un,

password = wrds_pw

)

on.exit(dbDisconnect(con), add = TRUE)

#function to get data

wrds_fetch <- function(sql) DBI::dbGetQuery(con, sql)

#importing data

ctff_features <- wrds_fetch("SELECT * FROM contrib_global_factor.ctff_features;")

ctff_chars    <- wrds_fetch("SELECT * FROM contrib_global_factor.ctff_chars;")

ctff_daily_ret <- wrds_fetch("SELECT * FROM contrib_global_factor.ctff_daily_ret;")

colnames(ctff_chars)

ctff_features[,]

library(data.table)

library(glmnet)

library(ranger)

set.seed(26)

#ensuring consistent types

dt0[, eom := as.Date(eom)]

dt0[, ctff_test := as.logical(ctff_test)]

dt0 <- dt0[!is.na(get(ycol))]

```

```

dt0[, (feature_cols) := lapply(.SD, as.numeric), .SDcols = feature_cols]

#choosing lambda

lambda_grid <- 10^seq(3, -3, length.out = 25)

#creating function to choose the best lambda

choose_lambda_once_fast <- function(dt_all, feature_cols, ycol, lambda_grid, calib_end) {

  train_all <- dt_all[eom <= calib_end]

  months <- sort(unique(train_all$eom))

  if (length(months) < 48) return(lambda_grid[round(length(lambda_grid)/2)])

  val_months <- tail(months, 24)

  tr_months <- setdiff(months, val_months)

  tr <- train_all[eom %in% tr_months]

  va <- train_all[eom %in% val_months]

  #standardizing on these subsets only

  tr_std <- standardize_panel(copy(tr), feature_cols)

  va_std <- standardize_panel(copy(va), feature_cols)

  X_tr <- as.matrix(tr_std[, ..feature_cols]); y_tr <- tr_std[[ycol]]

  X_va <- as.matrix(va_std[, ..feature_cols]); y_va <- va_std[[ycol]]

  fit <- glmnet(X_tr, y_tr, alpha = 0, lambda = lambda_grid, standardize = FALSE)

  pred_va <- predict(fit, X_va)

  mse <- colMeans((pred_va - y_va)^2)

  lambda_grid[which.min(mse)]

}

test_months <- sort(unique(dt0[ctff_test == TRUE, eom]))

first_test <- test_months[1]

```

```

all_months <- sort(unique(dt0$eom))

calib_end <- all_months[match(first_test, all_months) - 1]

#choosing and outputting the best lambda

best_lambda <- choose_lambda_once_fast(dt0, feature_cols, ycol, lambda_grid, calib_end)

cat("Chosen ridge lambda:", best_lambda, "\n")

#setting up for ensemble method - Ridge + Random Forest

ROLL_MONTHS <- 120

LONG_SHORT_FRAC <- 0.20

RETRAIN_EVERY <- 12

MAX_TRAIN_N <- 150000

NUM_TREES <- 300

W_RIDGE <- 0.7

rf_model <- NULL

out_list <- vector("list", length(test_months))

for (k in seq_along(test_months)) {

  t_month <- test_months[k]

  cat("Month:", as.character(t_month), "(", k, "/", length(test_months), ")\n")

  train_dt <- dt0[eom < t_month]

  pred_dt <- dt0[eom == t_month]

  if (nrow(pred_dt) < 50 || nrow(train_dt) < 5000) next

  #rolling window

  train_months <- sort(unique(train_dt$eom))

  if (length(train_months) > ROLL_MONTHS) {

    cutoff <- train_months[length(train_months) - ROLL_MONTHS + 1]

```

```

train_dt <- train_dt[eom >= cutoff]
}

if (nrow(train_dt) < 5000) next

train_dt <- standardize_panel(copy(train_dt), feature_cols)

pred_dt <- standardize_month_dt(copy(pred_dt), feature_cols)

#ridge prediction

X_train <- as.matrix(train_dt[, ..feature_cols])

y_train <- train_dt[[ycol]]

X_pred <- as.matrix(pred_dt[, ..feature_cols])

fit_ridge <- glmnet(X_train, y_train, alpha = 0, lambda = best_lambda, standardize = FALSE)

pred_ridge <- as.numeric(predict(fit_ridge, X_pred))

#random forest prediction

do_retrain <- is.null(rf_model) || ((k - 1) %% RETRAIN_EVERY == 0)

if (do_retrain) {

  cat(" Training RF...\n")

  ntr <- nrow(train_dt)

  if (ntr > MAX_TRAIN_N) {

    set.seed(1000 + k)

    idx <- sample.int(ntr, MAX_TRAIN_N)

    train_use <- train_dt[idx]

  } else {

    train_use <- train_dt

  }

  train_df <- data.frame(y = train_use[[ycol]], train_use[, ..feature_cols])

```

```

rf_model <- ranger(
  y ~ .,
  data = train_df,
  num.trees = NUM_TREES,
  mtry = floor(sqrt(length(feature_cols))),
  min.node.size = 50,
  sample.fraction = 0.7,
  seed = 123
)
}

pred_df <- data.frame(pred_dt[, ..feature_cols])
pred_rf <- as.numeric(predict(rf_model, data = pred_df)$predictions)

#ensemble method
pred_ens <- W_RIDGE * pred_ridge + (1 - W_RIDGE) * pred_rf
pred_dt <- copy(pred_dt)
pred_dt[, pred := pred_ens]

#weights
setorder(pred_dt, pred)
n <- nrow(pred_dt)
q <- max(1, floor(LONG_SHORT_FRAC * n))
pred_dt[, w := 0.0]
pred_dt[1:q, w := -1/q]
pred_dt[(n-q+1):n, w := 1/q]

```

```

    out_list[[k]] <- pred_dt[, .(id, eom, w)]
  }

output_df_ens <- rbindlist(out_list, use.names = TRUE, fill = TRUE)

#checking out weights (equal weights)

print(head(output_df_ens, 10))

#converting to data.table

dt_chars <- as.data.table(ctff_chars)

dt_weights <- as.data.table(output_df_ens)

dt_chars[, eom := as.Date(eom)]

dt_weights[, eom := as.Date(eom)]

#merging realised next-month excess return

dt_perf <- merge(

  dt_weights,

  dt_chars[, .(id, eom, ret_exc_lead1m)],

  by = c("id", "eom"),

  all.x = TRUE

)

#checking for missing realised returns

cat("Missing realised returns:", sum(is.na(dt_perf$ret_exc_lead1m)), "\n")

#monthly portfolio return calculation

port_ret <- dt_perf[,

,

.(port_ret = sum(w * ret_exc_lead1m, na.rm = TRUE)),

by = eom

```

```
]

setorder(port_ret, eom)

#computing Sharpe Ratio

mean_ret <- mean(port_ret$port_ret)

sd_ret <- sd(port_ret$port_ret)

sharpe <- mean_ret / sd_ret * sqrt(12)

cat("Mean monthly excess return:", mean_ret, "\n")

cat("Monthly volatility:", sd_ret, "\n")

cat("Annualized Sharpe ratio:", sharpe, "\n")
```