

Parsimonious Imputed Expanding Lasso

Common Task Framework

Author:	Lukas Jaeger
Affiliation:	Institute for Financial Management University of Bern
E-mail:	lukas.jaeger@unibe.ch
Submission Date:	16th February 2026

Preamble

The following procedure describes an algorithmic approach for building high-Sharpe-ratio portfolios using an extensive set of firm characteristics from Jensen et al. (2023). This description serves to document the code and methodology for my implementation of the Common Task Framework by Hellum et al. (2025).

Data Preparation

The first step involves preparing the data. A key challenge in financial panel data is the presence of missing data points. To address this issue, the algorithm uses Multivariate Imputation by Chained Equations (MICE) as proposed by van Buuren and Groothuis-Oudshoorn (2011). Rather than imputing each characteristic independently, MICE models each feature as a function of other observed characteristics within the same cross-section. This approach preserves the multivariate structure of the data and generates economically plausible imputations. The imputation is performed separately for each month to avoid any use of future information.

Within each month, every characteristic is first standardized cross-sectionally. When imputing a given feature, the MICE procedure estimates a regression using at most the 50 nearest predictors to reduce computational complexity. If a characteristic is entirely missing within a given month, the imputed value is set to zero in standardized units. A second standardization is performed post-imputation to ensure that the cross-sectional mean remains zero and the variance remains unit, as the imputation of missing values can subtly distort these statistical properties. Since imputation is performed independently across months, I parallelize the procedure at the month level to ensure computational tractability given the high dimensionality of the characteristic space.

Return Prediction

In the second stage, the algorithm uses the cleaned data to try and predict standardized stock returns. It uses the machine learning model Lasso (Least Absolute Shrinkage and Selection Operator) by Tibshirani (2018). The dependent variable is the cross-sectionally standardised one-month-ahead excess return. Standardizing the one-month-ahead returns ensures that my target variable represents relative performance. Since we are specifically interested in identifying stocks that outperform their peers, this transformation removes time-varying market volatility. The main reason I choose Lasso over a regular regression (or Ridge regression) is that Lasso is great at variable selection. Since we have 402 possible features, many of them might just be noise. Lasso has a built-in penalty that can set the influence of unhelpful features to zero.

The Lasso model solves the following optimization problem to find the best coefficients (β):

$$\hat{\beta}^{\text{Lasso}} = \arg \min_{\beta} \left(\frac{1}{T} (y - X\beta)'(y - X\beta) + \alpha \sum_j |\beta_j| \right) \quad (1)$$

where the first term represents the scaled sum of squared residuals, which measures the discrepancy between the observed returns (y) and the model predictions ($X\beta$). The second term, $\alpha \sum |\beta_j|$, is the L_1 penalty. The tuning parameter α controls the level of regularization; a higher α increases the sparsity of the model, forcing less significant coefficients to exactly zero and effectively performing variable selection.

To determine the optimal penalty parameter α , the algorithm implements an expanding window validation strategy. It begins with an initial training window of 60 months and evaluate performance on the subsequent 12 months. The window is then expanded forward in 12-month increments, each time estimating the model on the available training data and evaluating out-of-sample performance on the

next validation block. For each candidate α , it computes the mean squared error (MSE) on the validation sample and average performance across all validation windows. The value of α minimizing average validation MSE is selected. Finally, the model is re-estimated on the full training sample using the chosen hyperparameter.

Given the selected penalty parameter, the Lasso model including the tuned hyperparameter α is used to generate out-of-sample predictions of cross-sectionally standardized one-month-ahead excess returns for each stock-month observation in the test sample. These predicted standardized returns serve as the signal for portfolio construction.

Portfolio Creation

The final stage is turning those signals into an investment strategy. To do this, the algorithm ranks stocks from highest predicted standardized return to lowest and assigns them into deciles where the top decile includes stocks with the highest standardized return predictions. It constructs a long-short portfolio by taking an equal-weight long position in the top decile and an equal-weight short position in the bottom decile. Stocks in the remaining deciles receive zero weight. Within each leg, weights are normalized such that the long positions sum to +1 and the short positions sum to -1, yielding a dollar-neutral portfolio that is rebalanced monthly.

References

- Hellum, O., Jensen, T. I., Kelly, B. T., & Pedersen, L. H. (2025). The Power of the Common Task Framework. *URL: [https://ssrn.com/abstract, 5242901](https://ssrn.com/abstract/5242901)*.
- Jensen, T. I., Kelly, B., & Pedersen, L. H. (2023). Is There a Replication Crisis in Finance? *The Journal of Finance*, 78(5), 2465–2518.
- Tibshirani, R. (2018). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267–288.
- van Buuren, S., & Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45(3), 1–67.

Appendix

The resulting portfolio weights and the code is available for download via the following [link](#) and is also included below.

```

1 #####
2 """
3 PARSIMONIUS IMPUTED EXPANDING LASSO
4 """
5 #####
6
7 # Setup local environment and sync dependencies
8 !uv init
9 !uv add pandas sqlalchemy psycpg2-binary keyring pyarrow tqdm scikit-learn numpy
   ipywidgets joblib
10
11 # Imports
12 import time
13 import os
14 from pathlib import Path
15 import keyring
16 import pandas as pd
17 from sqlalchemy import create_engine, text
18 from sqlalchemy.engine import URL
19 import pyarrow
20 import numpy as np
21 from tqdm.auto import tqdm
22 from sklearn.experimental import enable_iterative_imputer
23 from sklearn.impute import IterativeImputer
24 from sklearn.linear_model import Lasso
25 from sklearn.metrics import mean_squared_error
26 from sklearn.exceptions import ConvergenceWarning
27 from joblib import Parallel, delayed
28 import warnings
29
30 # Functions
31
32 def prepare_data(chars: pd.DataFrame, features: pd.Series, eom: str) -> pd.DataFrame:
33     """
34     Parallel month-by-month standardization and MICE imputation of firm characteristics.
35
36     Args:
37         chars: Stock characteristics.
38         features: Series of feature column names.
39         eom: String name of the date column.
40
41     Returns:
42         DataFrame: A processed version of chars with features standardized (mean 0, std
43         1)
44                     and missing values imputed via IterativeImputer (MICE).
45     """
46     n_jobs = max(1, (os.cpu_count() or 1) - 4)
47
48     feat = features
49     characteristics = chars.copy()
50     characteristics[eom] = pd.to_datetime(characteristics[eom])
51
52     characteristics = characteristics.sort_values([eom, "id"]).reset_index(drop=True)
53     characteristics[feat] = characteristics[feat].astype("float64")
54
55     def _impute_month(g):
56         with warnings.catch_warnings():
57             warnings.filterwarnings("ignore", category=ConvergenceWarning)
58             warnings.filterwarnings("ignore", message="Skipping features.*")

```

```

58         mean = g[feat].mean()
59         std = g[feat].std().replace(0.0, 1.0)
60         X = (g[feat] - mean) / std
61
62
63         cols_with_data = X.columns[X.notna().any()].tolist()
64         if cols_with_data:
65             imputer = IterativeImputer(
66                 max_iter=5,
67                 random_state=42,
68                 sample_posterior=False,
69                 initial_strategy="median",
70                 skip_complete=True,
71                 n_nearest_features=50,
72             )
73             X.loc[:, cols_with_data] = imputer.fit_transform(X[cols_with_data])
74
75         X = X.fillna(0.0)
76         X = X - X.mean()
77         X = X / X.std().replace(0.0, 1.0)
78
79         g.loc[:, feat] = X
80         return g
81
82     with warnings.catch_warnings():
83         warnings.filterwarnings(
84             "ignore",
85             message="A worker stopped while some jobs were given to the executor",
86             category=UserWarning,
87         )
88
89         out = Parallel(
90             n_jobs=n_jobs,
91             backend="loky",
92             max_nbytes="10M"
93         )(
94             delayed(_impute_month)(g)
95             for _, g in tqdm(characteristics.groupby(eom),
96                             desc=f"Parallel Imputation ({n_jobs} cores)")
97         )
98
99         characteristics = (
100             pd.concat(out)
101             .sort_values([eom, "id"])
102             .reset_index(drop=True)
103         )
104
105     return characteristics
106
107 def lasso(train: pd.DataFrame, features: pd.Series, eom: str) -> Lasso:
108     """
109     Train a Lasso model using an expanding training set approach to optimize
110     the regularization parameter (alpha).
111
112     Args:
113         train: Training set containing stock characteristics and excess returns.
114         features: Series of feature column names.
115         eom: String name of the date column.
116

```

```

117 Returns:
118     Lasso: The final Lasso model fitted on the full training set
119           using the best alpha identified during walk-forward validation.
120 """
121 y_col = "ret_exc_lead1m"
122 train = train.copy()
123 feat = features
124
125 train[y_col] = train.groupby(eom)[y_col].transform(
126     lambda x: (x - x.mean()) / (x.std() if x.std() > 0 else 1.0)
127 )
128
129 all_months = sorted(train[eom].unique())
130 alphas = np.logspace(-3, 0, 50)
131
132 initial_train_size = 60
133 validation_step = 12
134 alpha_performance = {a: [] for a in alphas}
135
136 window_range = range(initial_train_size, len(all_months) - validation_step,
137                       validation_step)
138
139 for i in tqdm(window_range, desc="Expanding Training Set for Hyperparameter"):
140     tr_df = train[train[eom].isin(all_months[:i])]
141     va_df = train[train[eom].isin(all_months[i : i + validation_step])]
142
143     X_tr, y_tr = tr_df[feat].values, tr_df[y_col].values
144     X_va, y_va = va_df[feat].values, va_df[y_col].values
145
146     for a in alphas:
147         model = Lasso(alpha=a, max_iter=5000, random_state=42)
148         model.fit(X_tr, y_tr)
149         mse = mean_squared_error(y_va, model.predict(X_va))
150         alpha_performance[a].append(mse)
151
152 avg_mse = {a: np.mean(scores) for a, scores in alpha_performance.items()}
153
154 best_alpha = min(avg_mse, key=avg_mse.get)
155
156 X_full, y_full = train[feat].values, train[y_col].values
157 final_model = Lasso(alpha=best_alpha, fit_intercept=True, max_iter=10000,
158                     random_state=42)
159 final_model.fit(X_full, y_full)
160
161 return final_model
162
163 def predict_returns(model: Lasso, test: pd.DataFrame, features: pd.Series, eom: str) ->
164     pd.DataFrame:
165     """
166     Predict standardized returns using the trained Lasso model.
167
168     Args:
169         model: The fitted Lasso model from the lasso() function.
170         test: The test set DataFrame.
171         features: Series of feature column names.
172         eom: String name of the date column (e.g., 'eom').
173
174     Returns:
175         DataFrame: Contains columns 'id', 'eom', and 'pred_lead1m' with

```

```

173         the out-of-sample predictions.
174     """
175     test = test.copy()
176     feat = features
177
178     test = test.sort_values([eom, "id"]).reset_index(drop=True)
179     X = test[feat].to_numpy()
180
181     preds = model.predict(X)
182
183     lasso_predictions = test[["id", eom]].copy()
184     lasso_predictions["pred_lead1m"] = preds.astype(float)
185     return lasso_predictions
186
187 def create_ls_pf_w(pred: pd.DataFrame, test: pd.DataFrame) -> pd.DataFrame:
188     """
189     Build long-short portfolio weights based on predictions using a decile-based
190     equal-weighting strategy.
191
192     Args:
193         pred: DataFrame containing predictions.
194         test: The test set DataFrame containing stock identifiers and dates.
195
196     Returns:
197         DataFrame: A clean table with columns ['id', 'eom', 'w'], where 'w'
198                     represents the assigned portfolio weight (Long-Short).
199     """
200     n_deciles = 10
201
202     pf_creation = pred.merge(test[["id", "eom"]], on=["id", "eom"])
203
204     pf_creation['decile'] = pf_creation.groupby('eom')['pred_lead1m'].transform(
205         lambda x: pd.qcut(x, n_deciles, labels=False, duplicates='drop')
206     )
207
208     pf_creation['is_long'] = (pf_creation['decile'] == n_deciles - 1).astype(int)
209     pf_creation['is_short'] = (pf_creation['decile'] == 0).astype(int)
210
211     counts = pf_creation.groupby('eom')[['is_long', 'is_short']].transform('sum')
212
213     pf_creation['w'] = 0.0
214     pf_creation.loc[pf_creation['is_long'] == 1, 'w'] = 1.0 / counts['is_long']
215     pf_creation.loc[pf_creation['is_short'] == 1, 'w'] = -1.0 / counts['is_short']
216
217     portfolio_weights = pf_creation[["id", "eom", "w"]].copy()
218
219     return portfolio_weights
220
221 def main(chars: pd.DataFrame, features: pd.DataFrame, eom: str) -> pd.DataFrame:
222     """
223     Parsimonious Imputed Expanding Lasso:
224     Main function to prepare data, train model, predict standardized returns,
225     and calculate portfolio weights.
226
227     Args:
228         chars: Stock characteristics (ctff_chars.parquet).
229         features: Computed features (ctff_features.parquet).
230         eom: String name of the date column (default 'eom').
231

```

```

232 Returns:
233     DataFrame: Final portfolio weights with columns ['id', 'eom', 'w'].
234
235     """
236     eom = 'eom'
237     characteristics = chars.copy()
238     characteristics[eom] = pd.to_datetime(characteristics[eom])
239     feat_list = features.tolist() if isinstance(features, pd.Series) else list(features)
240
241     characteristics_full = prepare_data(characteristics, feat_list, 'eom')
242
243     train = characteristics_full[characteristics_full['ctff_test'] == False]
244     test = characteristics_full[characteristics_full['ctff_test'] == True].copy()
245
246     model = lasso(train, feat_list, 'eom')
247     lasso_pred = predict_returns(model, test, feat_list, 'eom')
248     pf_weights = create_ls_pf_w(lasso_pred, test)
249
250     return pf_weights

```