DEEP INCREMENTAL LEARNING FOR FINANCIAL TEMPORAL TABULAR DATASETS WITH DISTRIBUTION SHIFTS

Thomas Wong
Imperial College London
London
mw4315@ic.ac.uk

Mauricio Barahona
Imperial College London
London
m.barahona@imperial.ac.uk

ABSTRACT

We present a robust deep incremental learning framework for regression-based ranking tasks on financial temporal tabular datasets which is built upon the incremental use of commonly available tabular and time series prediction models to adapt to distributional shifts typical of financial datasets. The framework uses a simple basic building block (decision trees) to build hierarchical models of any required complexity to deliver robust performance under adverse situations such as regime changes, fat-tailed distributions, and low signal-to-noise ratios. As a detailed study, we demonstrate our scheme using XGBoost models trained on the Numerai dataset and show that a two layer deep ensemble of XGBoost models over different model snapshots delivers high quality predictions under different market regimes. We also show that the performance of XGBoost models with different number of boosting rounds in three scenarios (small, standard and large) is monotonically increasing with respect to model size and converges towards the generalisation upper bound. We also evaluate the robustness of the model under variability of different hyperparameters, such as model complexity and data sampling settings. Our model has low hardware requirements as no specialised neural architectures are used and each base model can be independently trained in parallel.

***K*eywords** Machine Learning, Time Series Prediction, Deep Learning,

1 Introduction

Many important applications of machine learning (ML), such as the Internet of Things (IoT) [1] and cyber-security [2], involve data streams, where data is regularly updated and predictions are made *point-in-time*. Such applications pose challenges to standard ML approaches, specifically with regard to the balance between model learning and their update in response to new data arrivals [3].

Incremental learning (IL) techniques [4, 5, 6] are used to adapt deployed machine learning systems to changes in data streams. For example, in image classification systems, class incremental learning [7, 8, 9] is used where the categories of images cannot be known in advance. A key challenge in IL is the presence of distributional shifts in data (or concept drifts) [10] which results in model degradation [11] during inference, i.e., deterioration of out-of-sample performance when the model learns relationships from the training set that significantly differ from those in the test set.

Reinforcement Learning (RL) [12, 13, 14] provides an alternative approach to prediction tasks in systems under data innovation. In RL, a model (agent) learns a policy to optimise its reward by interacting and eliciting a response from the environment. RL is therefore useful when the actions of the model influence the environment, and when multiple agents interact with each other [15]. However, if the actions of models have no influence on the data stream, (i.e., there is no feedback between agent and environment), then RL reduces to incremental learning. Furthermore, applying trained RL agents to unknown situations (e.g., trading [16], self-driving cars [17], or robotics [18]) remains a challenge, and complex algorithms have been introduced to bridge the gap between controlled environments and real-life situations [19, 20]. Hence the applicability of RL models can suffer from lack of robustness[21] and interpretability [22] of agent behaviour, and from the large amount of computational resources required.

The deep incremental learning (DIL) framework introduced here is a hybrid approach which allows predictions from base learner models to be reused in future predictions for tasks on data streams. Unlike RL, deep incremental learning only allows a single direction of information flow, from one layer to the next. Importantly, the *point-in-time* nature of predictions is preserved so that no look-ahead bias is introduced. DIL can be thought of as an extension of model stacking [23] but taking into account the stream nature of the data. Here, we consider an incremental problem in finance, which consists of ranking stocks for neutral portfolio optimisation applied to obfuscated data streams of tabular features and targets, corresponding to stocks and computed features. Such data sets are affected by strong non-stationarity and distribution shifts caused by regime changes in the market. Here, we expand on our previous work [24] and develop an IL framework that uses different data and feature sampling schemes and deep incremental model ensemble techniques appropriate for data streams with a high level of concept drift and non-stationarity.

Adopting an IL approach is crucial for data streams, as traditional assumptions of machine learning algorithms are not applicable. For instance, single-pass cross-validation that splits the data into *fixed* training, validation and test periods is not suitable. Under an IL framework, a model is represented by a continuous stream of parameters. Further, the procedure adds new hyperparameters to the model, such as training size and retrain period, which have a non-negligible impact on prediction performances for non-stationary datasets [25]. The distinction between features, targets and predictions is also blurred in the IL setting, as predictions from models learnt from different spans of data can be used as additional features when building other models, and targets can be created by subtracting against the predictions made. Therefore, model training is a *multi-step* problem, rather than a single-step problem. For an illustration of these issues, see Fig. 1.

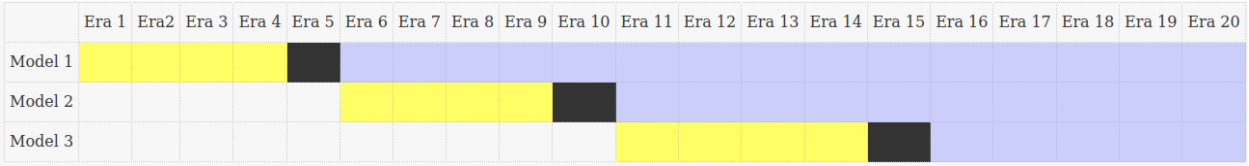


Figure 1: Schematic of how model predictions are reused in an incremental learning model. Consider three models (Models 1-3) each trained over a training period of 4 eras (weeks) and with a lag of 1 era (week). Model 1 is trained using information (both features and targets) up to Week 4 and after the 1-week lag, predictions are obtained for era 6 onwards. The features from Weeks 6-9 are combined with predictions from Model 1 to train Model 2. Similarly, Model 3 is trained using data from eras 11 to 14, plus predictions from Models 1 and 2.

Reusing model predictions within an IL framework provides a natural hierarchical structure, in which successive models can be interpreted as an improvement of previous ones in response to distributional shifts in data—this is akin to a feedback learning loop where model predictions correct themselves incrementally. Importantly, the prediction quality of each of the models can be inspected independently. Further, the IL setting allows models to process data streams with a finite memory usage by fixing the number of previous models that can be used by a model, so that the size of the training set (consisting of new features and predictions of previous models) remains bounded. There are many other possibilities for the design of IL models to deal with concept drifts in data. See [3, 10] for a survey on recent methods in modelling data streams with different change detection and adaptive learning techniques.

Our framework applies this hierarchical IL setting to a *collection* of machine learning models in parallel, which can be thought of as layers of models. However, in contrast to standard neural network architectures, such as the multilayer perceptron (MLP), the training is done in a single forward pass without back-propagation. This approach allows us to train complex model with reduced computational resources, as there is no need to put the whole model in distributed memory to pass gradients between layers. In this way, each model within a layer can be trained independently, and training becomes parallelised across GPUs without the need for specialised software packages to distribute data between GPUs. Recent work in deep learning suggests that backpropagation is not strictly necessary for model training [26]. For instance, Deep Regression Ensemble (DRE) [27] is built by training layers of ridge regression models with random feature projections. The deep incremental model presented here, on the other hand, focuses on data streams and temporal data and imposes no restrictions on the ML models used as building blocks forming the layers.

2 Temporal data formulations

Our work deals with prediction tasks motivated by financial temporal data streams, whereby the ranking of a group of stocks needs to be predicted based on the information available at era i. Such temporal data streams are treated under different formulations.

2.1 Temporal Tabular Datasets

Our temporal data is compiled into temporal tabular datasets, whereby the data at each time point is represented by features that have been computed from the time series up to that time.

Definition (Temporal Tabular Dataset)**.** A temporal tabular dataset is a set of matrices $\{X_i, y_i\}_{1 \leq i \leq T}$ collected over time eras 1 to T. Each matrix X_i represents data available at era i with dimension $N_i \times M$, where N_i is the number of samples in era i and M is the number of features describing the samples. The y_i are the targets to be predicted from the features X_i, and can be single-dimensional or multi-dimensional. The definition of the features is fixed throughout the eras, in the sense that the same computation is used to obtain the same number of features M at each era. Although the features can be in different formats (i.e., numerical, ordinal or categorical), they are usually transformed into equal-sized or Gaussian-binned numerical (ordinal) values. Note that the number of data samples N_i does not have to be constant across time.

Remark (Data Lag)**.** Unlike standard online learning problems, where newly arrived data are used immediately to generate predictions and to update the models, in financial applications there is usually a fixed time lag for the targets from an era to become known (also known as *data embargo*). If the data embargo is, e.g., equal to 5 eras, the targets of era t become known at era $t + 5$, and only then can they be used to calculate the quality of predictions according to a suitably chosen metric.

2.2 Time Series Data

In contrast, many traditional methods use time series directly to infer models for prediction.

Definition (Multivariate time series)**.** A multivariate time series of T steps and N channels can be represented as a matrix $\mathcal{X}_T = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{N \times T}$, where $1 \leq i \leq T$ and each (column) vector $\boldsymbol{x}_i \in \mathbb{R}^N$ contains the values of the N channels at time i. In many applications, the number of channels N is assumed to be fixed throughout time, with regular and synchronous sampling, i.e. the values in each vector from the N channels arrive simultaneously at a fixed frequency.

Although here we will concentrate on methods to predict temporal tabular datasets, there is a large variety of time series models that predict the time series directly.

Definition (Time Series Model)**.** Given a time series $\mathcal{X}_T \in \mathbb{R}^{\mathbb{N} \times \mathbb{T}}$, a (one-step ahead) time series model is a function $f : \mathbb{R}^{\mathbb{N} \times \mathbb{T}} \mapsto \mathbb{R}^N$ that predicts the vector $\boldsymbol{x}_{T+1}$ from X_T. In practice, the function f is often learned by training statistical or ML models using different instances of $\mathcal{X}_T$ obtained by shifting T across the time dimension.

A simple example of such a model, which will be used below, is the Exponential Moving Average (EMA). Moving averages are commonly used to capture trends in time series as follows.

Definition (Exponential Moving Average)**.** Given a univariate time series $x_1, x_2, \ldots, x_t, \ldots$, the exponential moving average of the time series at time t with decay α is defined as

$$y_t = (1 - \alpha) y_{t-1} + \alpha x_t \tag{1}$$

with initialisation $y_1 = x_1$.

Remark. More complex time series models have been developed, including sequence models in deep learning, such as LSTM [28] and Transformers [29]. However, these models tend to be overparameterised and lack robustness to regime changes [30]. They also involve heavy computational costs associated with the training and updating of models.

2.3 Transforming time series into temporal tabular datasets: feature extraction

There are a myriad of methods commonly used to transform multivariate time series into temporal tabular datasets. These feature engineering (FE) methods consist of feature extraction applied over a look-back window:

- Feature extraction: a function f that maps the time series $\mathcal{X}_T \in \mathbb{R}^{N \times T}$ to a feature space $f(\mathcal{X}_T) \in \mathbb{R}^M$ where M is the number of features. Feature extraction methods can help reduce the dimension and noise in time series data.
- Look-back window: Feature extraction is applied to data within a look-back window (memory) of fixed length k. Multiple look-back windows can also be used to extract features that capture short-term and long-term trends, and concatenated to represent the state of the time series.

In this paper, we will employ two feature engineering methods that have been proposed for financial time series:

- *Signature Transform (ST)*: STs [31, 32, 33] are deterministic transformations, recently proposed by Lyons, which can extract features at increasing orders of complexity from multivariate data, including time series. See [33] for a review of different applications of signature transforms in machine learning. For details on how STs are applied to the Numerai dataset, see Section 9.1 in the Supplementary Information.
- *Random Fourier Transform (RFT)*: RFTs have been used in [34] to model the return of financial price time series but can also be applied to extract features from time series at each time step. The key idea is to approximate a mixture model of Gaussian kernels with trigonometric functions [35]. Details on how RFTs are applied on the Numerai dataset are given in the Supplementary Information, see Algorithm 12 in Section 9.1.

Remark. As discussed in Section 3.2 in more detail, once feature extraction methods have been applied and temporal tabular datasets generated, traditional ML models such as ridge regression, gradient-boosting decision trees (GBDTs), and multi-layer perceptron (MLP) networks can be used to carry out predictions point-wise in time [24], without relying on complex and expensive advanced neural network architectures such as Recurrent Neural Networks (RNN), Long-Short-Term-Memory (LSTM) Networks or Transformers [29].

3 Machine learning for temporal data

Before describing our deep incremental learning approach, we give some relevant background and brief links to standard methods used for prediction of temporal tabular data. These methods will be used as the building blocks of our incremental learning approach.

3.1 Prediction of Temporal Tabular Datasets from time series data: Factor-timing models

Factor-timing models [36] are a well-used approach to produce predictions for a temporal tabular dataset from time series, whereby the raw predicted values from a time series model (e.g., the EMA (1)) are converted into normalised rankings, which are then used as weights for the linear factor-timing model (see Algorithm 1). As baseline for comparison, we apply below factor-timing models to time series that are derived from temporal tabular datasets through a transformation, as follows.

Definition (Derived Time Series)**.** A transformation f is applied to the tabular features X_t and targets y_t at era t to generate a multivariate time series: $\boldsymbol{\chi}_t = f(X_t, y_t)$, where $f : (\mathbb{R}^{N_i \times M}, \mathbb{R}^{N_i \times 1}) \mapsto \mathbb{R}^M$. For example, f can be the Pearson correlation between feature and targets.

This procedure generates a time series of *feature performances* from the temporal tabular dataset, which can be used within a factor-timing model, as in Algorithm 1 avoiding look-ahead bias.

Algorithm 1: Factor Timing Model

Input: At era t: predicted values $\hat{y}_t \in \mathbb{R}^M$ from a time series model, and temporal tabular dataset $X_t \in \mathbb{R}^{N_t \times M}$ where M is the number of features

Output: Factor-timing model predictions $\hat{z}_t \in \mathbb{R}^{N_t}$

Calculate normalised ranking of features $\hat{r}_t$ from predictions of time series model

$$\hat{r}_t = \text{rank}(\hat{y}_t) - 0.5,$$

where the rank function calculates the percentile rank of a value within a vector, so that $-0.5 \leq \hat{r}_t \leq 0.5$.

If needed, given upper bound u and lower bound l $-0.5 < l < u < 0.5$, apply truncation to $\hat{r}_t$:

$$r_t = \max(\min(\hat{r}_t, u), l).$$

Calculate linear factor-timing predictions $\hat{z}_t = X_t r_t$

3.2 Machine Learning Models for Temporal Tabular Dataset prediction

In contrast to factor-timing models, ML methods can be applied directly to temporal tabular datasets for prediction tasks. There is a rich literature comparing different machine learning approaches on tabular datasets [37, 38, 39, 40]. Several benchmarking studies [37, 38, 39] have demonstrated that advanced deep learning methods, such as transformers [41] and other neural network (NN) models , underperform for regression/classification tasks on tabular datasets relative to traditional approaches, such as GBDT or MLP models. In particular, recent research [37] has shown that GBDTs with moderate hyperparameter tuning perform closely to much more complex NN models.

Further, previous studies had focused on datasets with relatively small numbers of features and samples ($M < 200$ features, $N_i < 10,000$ data rows or samples), whereas we are interested in large datasets with more than 1000 features and more than 200,000 data rows. For larger datasets, it has been shown [37] that GBDTs performed better than 11 neural-network-based approaches and 5 other baseline approaches, such as Support Vector Machines. GBDT models also display higher performance when feature distributions are skewed or heavy-tailed.

Finally, our objective is the prediction of data streams that are not static or stationary, but rather dynamic and subject to distribution shifts. Previous work has shown that GBDTs and MLPs outperform other deep learning approaches for temporal tabular datasets, with higher robustness and lower computational requirements for training (and retraining) of models [24, 38, 39].

In this paper, GBDT models are studied in detail for tabular prediction, as it has demonstrated strong performances in benchmarking studies [37, 38, 39, 24] and there exist efficient implementations that allow scalable model training and inference.

Details of the GBDT and MLP models can be found in section and in the Supplementary Information.

4 Deep (hierarchical) Incremental Learning algorithm for temporal data

Our deep incremental learning model is built layer by layer, using component models of a given type (e.g., factor-timing models or GBDTs) composed hierarchically across layers, as follows. At any time, we split our temporal dataset into segments of temporal history. Each segment is assigned to a layer, and for each layer we train an ensemble of models computed with different random seeds. We thus define the number of layers L, the sizes of the training data ('lookback window') for each layer $(a_1, \ldots, a_L)$, and the number of models in the ensemble within each layer $(K_1, \ldots, K_L)$.

The models are learnt using information from different temporal segments sequentially and hierarchically, layer by layer, so that past predictions can be used to refine future predictions. Operationally, at the start of the training in a layer l, we prepare the features and targets $\{X_j^l, y_j^l\}$ that are shared by all K_l component models within the layer using the most recent data from the specified lookback window. Importantly, the features used as inputs to a layer consist of both original features from the temporal tabular dataset plus predictions obtained from models trained in previous layers $i = 1, \ldots, l-1$ (see Fig. 1).

Regarding the type of component models that form the ensemble in each layer, any model that uses tabular features as input and predicts tabular targets can be used. This expands the class of models from standard tabular models, such as GBDT and MLP, to other multi-step models, such as factor-timing models. The overall model is therefore a composition of such component models.

Each component model within a layer is trained in an incremental manner. This means that the model parameters are updated at regular intervals as new data arrives only using the data from the given lookback window. Other hyperparameters of the model (e.g., boosting rounds for GBDTs) remain unchanged. For example, if the dataset in total has 1000 eras and we update the models every 50 eras with lookback window equal to 600 eras we would obtain 9 models, with model training at Eras 600,650,700, ..., 1000.

The component models within each layer can be trained in parallel, which allows the incremental learning model to be efficient and scalable.

The pseudocode in Algorithm 2 outlines the overall structure of the computational framework.

Algorithm 2: Deep IL model with model stacking

Input: Temporal Tabular Dataset $\{X_i, y_i\}_{1 \leq i \leq T}$, number of layers L, the number of models within each layer $(K_1, \ldots, K_L)$, sizes of training data in each layer $(a_1, \ldots, a_L)$, data embargo b,

for $1 \leq l \leq L$ **do**

  Assign the temporal window w_l to layer l: $w_l = \{\sum_{w=1}^{l}(a_w + b) < j \leq \sum_{W=1}^{l-1}(a_w + b) + a_l\}$

  Prepare training data for layer l: $\{X_j^l, y_j^l\}_{j \in w_l}$, where the features X_j^l can be any combinations of predictions from previous layers and the original features in the temporal tabular dataset.

  for $1 \leq k \leq K_L$ **do**

    Perform data and feature sub-sampling for each component model $\mathcal{M}_k^l$

    Train component model $\mathcal{M}_k^l$ with regular updates

    Obtain predicted ranking of stocks using $\mathcal{M}_k^l$ from era $1 + \sum_{w=1}^{l}(a_w + b)$ onward to be used in model training in subsequent layers

  end

end

This framework leads to a deep hierarchical ensemble of models, where each layer takes advantage of model ensembling, and the integration of information across layers through functional composition enables the incremental learning necessary to adapt to non-stationarity and regime changes. We now discuss briefly some characteristics of the model:

Hierarchical nature of the model and self-similarity: The proposed framework is hierarchical: the ensemble of models in any given layer, which is used to generate predictions in time beyond the latest data arrival, integrates hierarchically both data and predictions obtained from the models in the preceding layers, themselves fitted to previous time periods. Indeed, the model has characteristics of self-similarity, since the layered structure can be seen as performing a functional composition of learning models of the same type, e.g., the component models within each layer can be chosen to be GBDTs (or MLPs) so that the learning mechanism of each individual component is similar to the overall model, and the structure is extended repeatedly in a self-similar manner by interpreting a component model as a base learner for another component model in a higher layer.

Universal Approximation Property: It is well known that MLP and GBDT models have the universal function approximation property [42], and Deep Learning models for sequences, such as LSTM [43], also have the universal function approximation property for any dynamical system. Since the DIL model is a composition of models each of which has the universal function approximation property, it also has the universal approximation property for the underlying stochastic process that drives the data generation of the temporal tabular dataset.

Model stacking: bagging and boosting across time Our model can also be interpreted as a *stacked model* with a total of $\sum_{i=1}^{L} K_i$ base learners, such that K_i base learners are trained in the i-th iteration, corresponding to each of the L layers. Ideas from bagging and boosting are integrated within the model. Each layer consists of multiple models trained in parallel, as in bagging, so that variance is reduced by combining predictions from different models within a layer. Further, our model can be considered as a degenerate case of boosting, where the learning rate of the target is set to zero inter-layers, such that the target is not adjusted based on predictions from previous layers. However, the architecture can be modified to allow for target adjustment (boosting) between layers if needed.

Adaptive nature of the model A key characteristic of the DIL model is that it is designed to support *dynamic* model training, with parameters of each component model updated regularly to adapt to distributional shifts in data. Under the traditional machine learning framework, hyperparameters are selected by cross-validation on splits of the training data. Yet optimal hyperparameters based on a single test period might not work in future. In the DIL model, predictions from previous layers based on different model hyperparameters are combined in the successive layer, corresponding to a later span of time, acting as a dynamic soft selection of hyperparameters. It has been shown that stacking of models with different random seeds [44], hyperparameters [45] and architectures [46] leads to robust performance for *static* datasets. The DIL model can thus be seen as an extension of stacking techniques to *stream* datasets, so that models incrementally trained with incoming data streams are stacked to obtain more robust predictions.

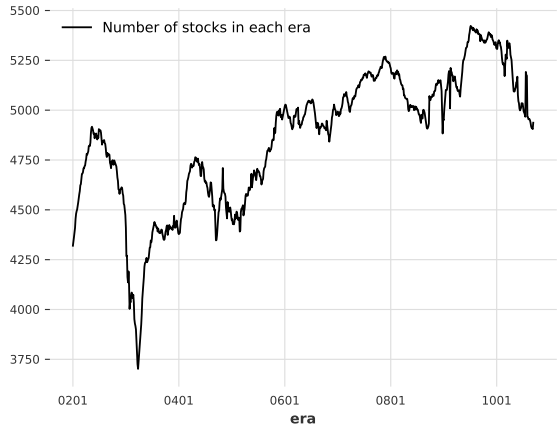


Figure 2: Number of stocks in each era from Era 201 to Era 1070 for v4.2 Numerai dataset.

5 Prediction tasks for neutral portfolio optimisation using financial data from the Numerai competition

Numerai dataset and prediction task As discussed above, financial time series data can be used directly for prediction [47, 29], yet such methods tend to be overfitted, making them less robust to regime changes and to the high stochasticity inherent to financial data. Alternatively, feature engineering is applied at each era to compute features that capture different aspects of the time history over look-back periods. This approach leads to a temporal tabular dataset, which can be used for prediction without considering time explicitly. The Numerai competition is based on one such professionally curated temporal tabular dataset, formed by matrices X_i that contain M stock market features (computed by Numerai) for N_i stocks updated weekly (i.e., eras are weeks). The definition and computation of the features is fixed throughout the eras. Importantly, the dataset is *obfuscated*, i.e., the identity of the stocks present each week is unknown. The task is then to predict the stock rankings each week, from lowest to highest expected return. This ranking is used to construct a market-neutral portfolio.

Features and Targets Two versions of the Numerai dataset, V4.1 (Sunshine) and V4.2 (Rain) [48, 49] are used in this study, starting on 2003-01-03 (Era 1) and extending up to 2023-06-30 (Era 1070) [1]. The dataset is weekly, i.e., eras correspond to weeks.

Each week, Numerai makes public a feature matrix of 1586 (V4.1)/ 2131 (V4.2) features for a changing selection of (unidentified) stocks, selected according to risk management rules by the Numerai hedge fund, plus several targets corresponding to stock returns normalised by different proprietary statistical methods. In Figure 2, the number of stocks in each week (era) from Era 201 to Era 1070 are shown, which demonstrates the number of stocks traded varied in each week.

The features are normalised into 5 equal-sized integer bins, from -2 to 2, so that the bins have zero mean. The targets are scaled between 0 and 1, and grouped into 5 bins (0, 0.25, 0.5, 0.75, 1.0) following a Gaussian-like distribution, and then subtracting 0.5 to make the bins zero-mean. For a more extended discussion of the Numerai dataset, including features and targets, see Ref. [24].

In V4.2 dataset, some features have completely missing values up to Era 251. In Figure 3, we show the number of features with completely missing values in each era between Era 1 and Era 300. In the first 100 eras, we have around 50% of features with completely missing values. Therefore, we train XGBoost models using data from Era 201 onwards to ensure less than 10% of features have completely missing values.

Each feature is now assigned to one or more groups. There are 10 feature groups in total, namely Intelligence, Charisma, Strength, Dexterity, Constitution, Wisdom, Agility, Serenity, Sunshine and Rain. For all feature groups except the last one (Rain), they represent features that behave similarly, as they are derived from similar data sources [49]. The Rain feature group consists of features that are created synthetically from features in other groups using information up to Era 585 [2].

[1] The data keeps updating every week

[2] Numerai suggests most features are derived by fitting weights to the time-series of other features.

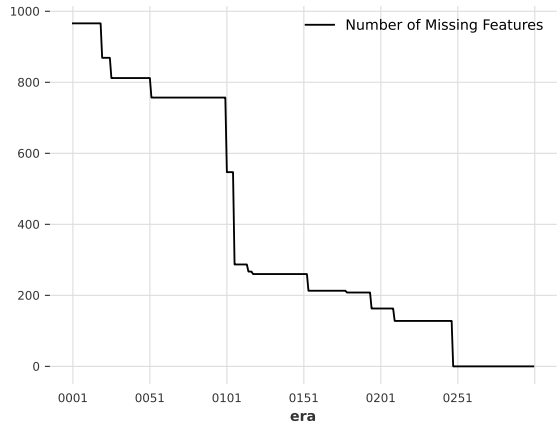


Figure 3: Number of features with missing values from Era 1 to Era 300 for v4.2 Numerai dataset.

Data Lag The data lag for predictions depends on the practicalities of the data pipeline. For Numerai, a lower bound for the scoring target to be resolved is 5 weeks (4 weeks of market data and 1 week for data processing). To take account into both the data lag for the data generation process from Numerai, and the time needed to train models, a conservative data lag of 15 weeks is used here.

Scoring Function Numerai calculates a variant of Pearson correlation for all predictions in a single era t, as follows [50]: Let y_p be the predictions ranked between 0 and 1, y_t the targets centred between -0.5 and 0.5, $\Phi(\cdot)$ the (cumulative) distribution function of a standard Gaussian, $\mathbf{sgn}(\cdot)$ and $\mathbf{abs}(\cdot)$ the element-wise sign and absolute value function, respectively, then the Numerai correlation score for era t, ρ_t, is given by:

$$\begin{aligned} y_g &= \Phi^{-1}(y_p) \\ y_{g15} &= \mathbf{sgn}(y_g) \cdot \mathbf{abs}(y_g)^{1.5} \\ y_{t15} &= \mathbf{sgn}(y_t) \cdot \mathbf{abs}(y_t)^{1.5} \\ \rho_t &= \mathbf{Corr}(y_{g15}, y_{t15}) \end{aligned}$$

where $\mathbf{Corr}(\cdot, \cdot)$ is the Pearson correlation function. Note that the 3/2 power is taken to emphasise the contribution from the highest and lowest predictions. The correlation score ρ_t is collected for each era t over the test period to calculate the following portfolio metrics:

- Mean Corr: average of ρ_t over all eras in the test period
- Maximum Drawdown: maximum difference between the cumulative peak (high watermark) and the cumulative sum of correlation scores in the test period
- Sharpe ratio: ratio of Mean corr and standard deviation of ρ_t over all eras in the test period
- Calmar ratio: ratio of Mean Corr and Maximum Drawdown

We will use these metrics to score our models throughout the paper. Specifically, high values of 'Mean Corr', 'Sharpe ratio' and 'Calmar ratio' are all indicative of good model performance. We use the main target decided by Numerai, 'target-cyrus-v4-20' for scoring the trained models.

Example of concept drift Regime changes is one of the reasons why machine learning trading strategies suffer from significant losses [<**empty citation**>]. Machine learning trading strategies learn historical patterns from a vast amount of financial data. When there are regime changes, these patterns become obsolete or even incorrect such that they are no longer able to predict the future return of financial assets.

Regime changes are often unpredictable. For example, considering the return from Numerai hedge fund [51], the risk-adjusted return of hedge fund from September 2019 up to March 2023 is spectacular, where the maximum drawdown is less than 5%. However, from March 2023 there are 4 consecutive months of negative returns, giving a cumulative drawdown more than 33%. Indeed, most risk-management metrics based on historical performances, such as Value-at-Risk (VaR) [52] would not be able to foresee this downturn.

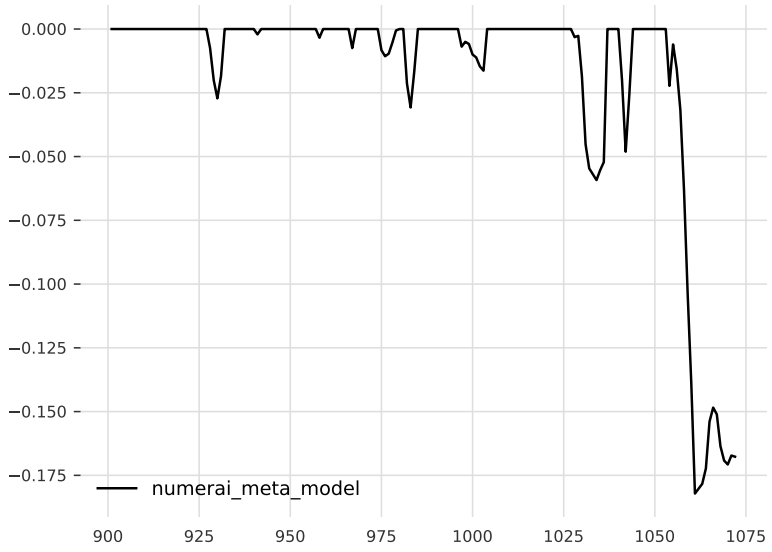


Figure 4: Underwater(Drawdown) plot of the Numerai Meta Model between Era 901 to Era 1070. A large drawdown is experienced by the model between Era 1055 to Era 1070.

The challenging period for Numerai hedge fund corresponds to Era 1055 to Era 1070 in the dataset. Similar to the hedge fund, predictions from models submitted by participants also suffer from a large drawdown in the same period. In Figure 4, we show the Underwater(Drawdown) plot of the Numerai Meta Model. The drawdown in Era 1055 is Era 1070 is around 4 times bigger than historical drawdown, suggesting there could be concept drift in the data.

Here, we define market regime **post hoc** based on performances. 2023-02-17 (Era 1051) to 2023-06-30 (Era 1070) is defined as the bear market. 2020-04-04 (Era 901) to 2023-02-10 (Era 1050) is defined as the bull market. In Table 1, we report the performances of the Numerai Meta Model from Era 901 to Era 1070 for the whole period and under both market regimes. Under bull market, we have a better than average performance while under bear market we have a negative performance. Over a long enough period, model predictions have a positive return but models can experience large drawdown in bear market, causing a lot of volatility to the portfolio.

Regime	Mean Corr	Sharpe	Calmar
All	0.0175	0.7915	0.0962
Bull	0.0207	1.0085	0.3491
Bear	-0.0062	-0.3220	-0.0341

Table 1: Performances of Numerai Meta Model from Era 901 to Era 1070 under different market regimes.

6 Incremental Learning for Numerai prediction: Non-hierarchical models

Before presenting results from our hierarchical (deep) incremental learning model, we develop non-hierarchical incremental learning models for the Numerai dataset. These types of models have been used in the literature [34, 53, 54], and will serve here both as a baseline comparison and to guide some our choices in model type, training methods and hyperparameter selection. Although these models are updated incrementally (i.e., they do incorporate information of new data arrivals) they do not incorporate information hierarchically across multiple layers, and hence fail to generalise well, due to severe distribution shifts in the data.

To enhance the breadth of our comparison, we study here two types of IL models: (i) factor-timing models that use explicit time series derived from the Numerai dataset, and (ii) ML algorithms (GBDTs, MLP) for tabular datasets which are used directly on the Numerai temporal tabular dataset.

6.1 Factor Timing Models

We generate three factor-timing (FT) models (based on Exponential Moving Average, Signature Transform, and Random Fourier Transform), all of which follow the setup in Algorithm 1 but are generated using specific transformations of the data, as follows.

We obtain a multivariate time series from the V4.2 dataset $\{\tilde{X}_t, y_t\}_{t=1}^{1070}$ as described in Section 3.1, i.e., we generate the time series $\{\boldsymbol{\chi}_t\}_{t=1}^{1070}$, where each $\boldsymbol{\chi}_t \in \mathbb{R}^{2132}$ is derived by computing the correlation between each feature and the target y_t.

Once the time series is computed, we train factor timing models at each era using all the available data up to that point, bar the data embargo of 15 eras.

We train the following FT models, one using an EMA model of the time series and two other using transforms that generate features from the time series:

- Exponential Moving Average factor-timing model: An EMA model (1) is computed for each of the 2132 feature series independently. This multivariate model is used to produced predictions $\hat{y}_t \in \mathbb{R}^{2132}$ for each era t, which are then used within the FT model to produce model predictions $\hat{z}_t \in \mathbb{R}^{N_t}$, as given by Algorithm 1. These predictions are then scored using our portfolio metrics.
- Feature Transform factor-timing models: From a random subset of the 2132 variables of the time series $\boldsymbol{\chi}_t$ we generate transformed features (ST or RFT) with lookback period using all available data. This process is repeated for a varying number of randomly drawn subsets of the variables in $\boldsymbol{\chi}_t$ to explore the importance of model complexity c, defined as the ratio of number of features and length of the time series (hyperparameter in Table **??**). For example, for a time series of length $T = 600$, we may wish to generate models with complexity $c = 2$. Therefore, we obtain $c \cdot T = 1200$ ST features by taking 60 subsets randomly sampled from $\boldsymbol{\chi}_t$, where each random subset of 4 time series generates 20 ST features (taking signatures up to level 2). An analogous procedure is followed for RFT.

  The $c \cdot T$ transformed features (ST/RFT) from all subsets are then concatenated, and ridge regression with L2-regularisation is applied to generate the linear model for $\hat{z}_t \in \mathbb{R}^{N_t}$.

  Following recent research in high dimensional ridgeless regression [55, 34], we average the results of ridge regression over a range of regularisation parameters (0.01,0.1,1.0,10.0,100.0) that cover a spectrum of models, from dense to sparse.

The FT models are retrained at every era using all data available up to that point, hence by construction these models are all incremental. In Algorithm 3, we describe the incremental learning procedure to train FT models.

Algorithm 3: Factor Timing Models

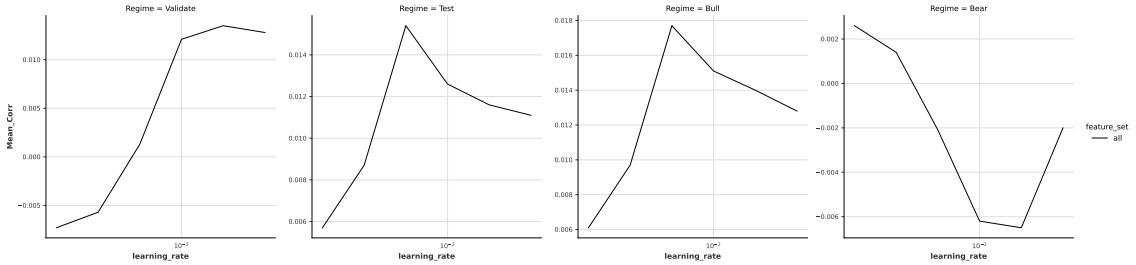
Input: Data embargo $b = 15$
for $401 \leq i \leq 1070$ **do**
  Calculate feature performances time series $\{\boldsymbol{\chi}_t\}_{t=1}^{1070}$, where each $\boldsymbol{\chi}_t \in \mathbb{R}^{2132}$ using the procedure described in Section 3.1
end
for $801 \leq i \leq 1070$ **do**
  Prepare training data by slicing the feature performances time series from Era 1 to $D_i - b$
  Train Factor Timing models, which can be one for EMA, Signature Transforms and Random Fourier Transforms model.
  Get one-step ahead prediction y_{i+1} for era D_{i+1}
  Create factor timing predictions z_{i+1} using Algorithm 1
end

To optimise key hyperparameters of the FT models (decay α for EMA models, and complexity c for ST/RFT models), we evaluate their one-step ahead performance over the *validation* period from Era 801 - Era 885.

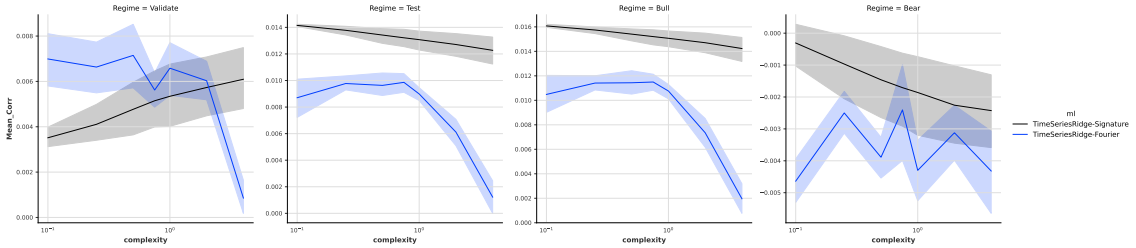
Figure 5(a) shows the performances of the EMA models with different weight decays under different market regimes. 6 different weight decays are considered: 0.00125,0.0025,0.005,0.01,0.02,0.04. The best weight decay for Mean Corr in validation period (0.02) is different from the best weight decay for Mean Corr in the test period (0.005). This shows the optimal weight decays are changing with respect to time and time-series cross-validation cannot always select the best weight decay for out-of-sample data. Models with a lower weight decay has a better performances in Bear market but the worst in Bull market, suggesting model behaviours vary under different market regimes.

Figure 5(b) shows the Mean Corr of the Feature Transform factor-timing models under different market regimes, averaged over 4 different random seeds. Results for other risk metrics, Sharpe ratio and Calmar ratio are shown in Figure 30 and 31 respectively.

We train Feature Transform factor-timing models with different complexities $c = 0.1, 0.25, 0.5, 0.75, 1, 2, 4$. RFT performs better than ST models in validation period for small model complexity ($c < 1$) but not in the test period. For RFT models, Mean Corr decreases as model complexities increases. For ST models, Mean Corr increases as model complexities increases in validation period but not in test period. It suggests using more complex factor-timing models does not always give better results for FT models. The best model hyperparameters selected based on time-series cross-validation is not robust in out-of-sample data.



(a) EMA models



(b) Feature Transform factor-timing models

Figure 5: Mean Corr of EMA and Feature Transform factor-timing models under different market regimes

6.2 Benchmark IL model for XGBoost models

Two key hyperparameters for IL models are (i) training size, which for a temporal tabular dataset, the number of eras of data to be used in training and (ii) retrain periods, which governs how often the model are retrained/updated using the latest data.

In this section, we train IL models with different training sizes and retrain periods, using XGBoost models with two different set of hyperparameters, one is the hyperparameters found by grid search with fixed number of boosting rounds $B = 5000$ and learning rate $L = 0.01$, which is called as the **Ansatz** hyperparameters set, the other is the hyperparameter set provided by Numerai in their example Python script, called as **Numerai** hyperparameters set. In each model retrain, we will use **all** the available data from Era 201 to train the models. For example, at Era 1000 which is the $6th$ retrain of model, we will use 800 eras of data from Era 201 to train the models. The first retrain at Era 801 will uses 600 eras of data, which is roughly equal to the training period of the Numerai example models using the first 12 years of data (≈ 574 eras).

For completeness, we also consider the situation where models are not retrained, which is called as **Ansatz-Fixed** and **Numerai-Fixed** respectively.

The details on the procedure to create hyperparameter sets and model training are described in Section 9.2 and 9.2.3 in SI. To speed up training, we train models using only around half of data in each era by removing observations with target equal to the Median value (0.5), and we show in SI Section 9.2.1 this sampling method does not significantly deteriorate model performances while able to reduce computational costs by half compared to training using all data in era.

We also regularly sample the data eras in the training period such that we only use 25% of data eras in model training. We then train 4 models each using 25% of data without overlap. For example, we use data from Era 1,5,9,... to train the first model. Similarly for the other 3 models. We call this procedure **regular era sampling**.

When we create benchmark models of size $B = 1000, 5000, 10000, 25000, 50000$. The train size of models are fixed to 600 (with the last 15 eras of data for embargo) with the start of training data at Era 201. For models with $B \leq 5000$, we regularly retrain models every 50th era, which corresponds to updating model once per year. We do not retrain models with $B \geq 10000$ due to computational reasons. The learning rates of the model are determined using the Ansatz formula $L = \frac{50}{B}$, which is justified in Section 9.2.2 in SI.

We report performances of the benchmark models from Era 801 to Era 1070 by the following regimes. The validation period is Era 801 to Era 885. The test period is between Era 901 and Era 1070, where 15 eras of data embargo are used. We also report performance based on market regimes within the test defined to understand if models behave differently under different regimes.

- Validation: Era 801 - Era 885
- Test: Era 901 - Era 1070
- Bull: Era 901 - Era 1050
- Bear: Era 901 - Era 1070

In Figure 6 we show performance of the benchmark models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$. A more detailed comparison for the benchmark models with $B = 1000, 5000$ is given in Figure **??** in SI.

Model performances for models with $B \geq 5000$ are not significantly different in both validation and test period. Within different market regimes in the test period, there are also no significant differences in model performances. There is little improvement of model performances for using models with $B \geq 10000$.

For models with size $B = 1000, 5000$, there are no significant differences between each pair of models using the two different hyperparameter sets **Ansatz**, **Numerai** for a fixed data sampling method (regular retrain vs no retrain) in both the validation and test period.

As number of boosting rounds increases, the performance differences between models with different hyperparameters narrows. We suggest the differences between models performances are more due to differences between data sampling schemes used and to a lesser extent due to the hyperparameters.

We then select the benchmark model hyperparameters considering both model performances and amount of computational resources.

Despite **Numerai** models have a slightly better Mean Corr than **Ansatz** models, it has a higher computational time and memory costs (explained in detail in SI Section 9.2) and the improvement in Mean Corr is not significant in the validation period.

Models with **Ansatz** hyperparameters also have a lower variance than that with **Numerai** hyperparameters across different risk metrics, Mean Corr and Sharpe ratio.

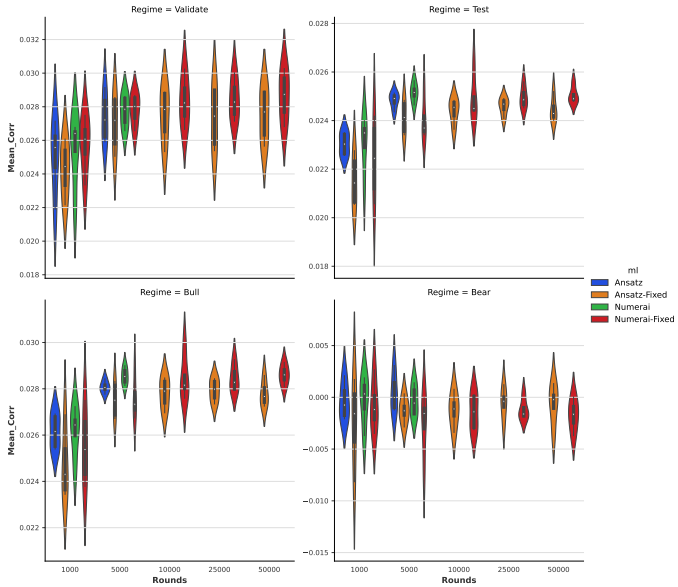
Given both set of benchmark models are very similar, we select the one with less computational resources, which is **Ansatz** hyperparameters to train different deep IL models in the next section.

How to measure similarity between two GBDT models To measure the similarity between two GBDT models, we can consider the overall structure similarity between two models using feature importance, which counts how many times a feature is used in a decision rule for a node within one of the trees in the GBDT model. It is not enough to consider correlation between predictions only because predictions that are similar by itself but based on different decision rules can still offer diversification benefits to the ensemble by providing different learning pathways. If multiple independent learners arrive at similar predictions based on different information, the prediction becomes more robust as even small drifts in the test data will not completely distort the picture.

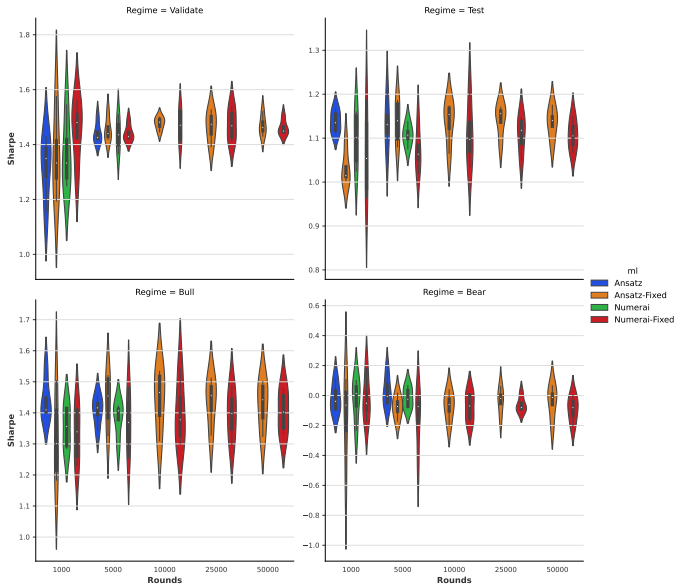
For simplicity, a correlation based measure is used to measure the overall similarity of two GBDT models. Note that this measure considers the averaged contribution of each feature towards the model and ignores the interaction between features.

Definition (Structural Similarity of GBDT models)**.** Given two GBDT models A and B with the same number of features M, let $R_A, R_B \in \mathbb{R}_+^{\mathbb{M}}$ be the feature importance of the models, the structural similarity $\mathcal{S}(A, B) \in [-1, 1]$ between two GBDT models is defined as the correlation between the normalised feature importance of the two models.

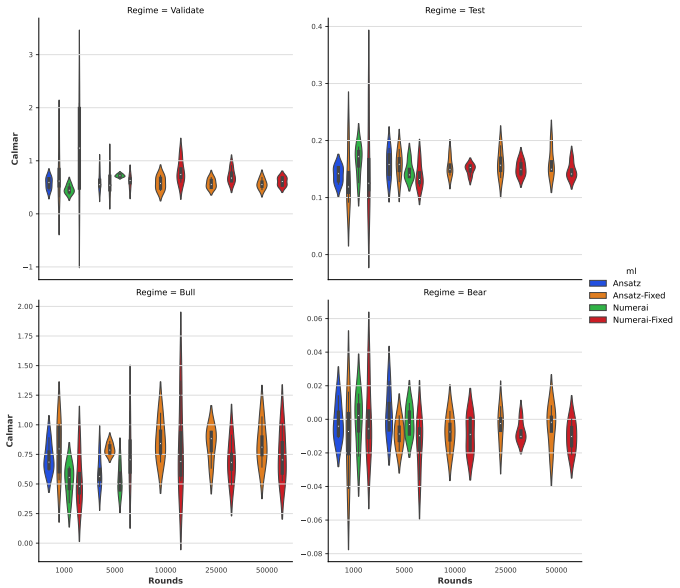
$$
\begin{aligned}
r_A &= \text{rank}(R_A) \in [0, 1]^M \\
r_B &= \text{rank}(R_B) \in [0, 1]^M \\
\mathcal{S}(A, B) &= \textbf{Corr}(r_A, r_B)
\end{aligned}
$$



(a) Mean Corr



(b) Sharpe



(c) Calmar

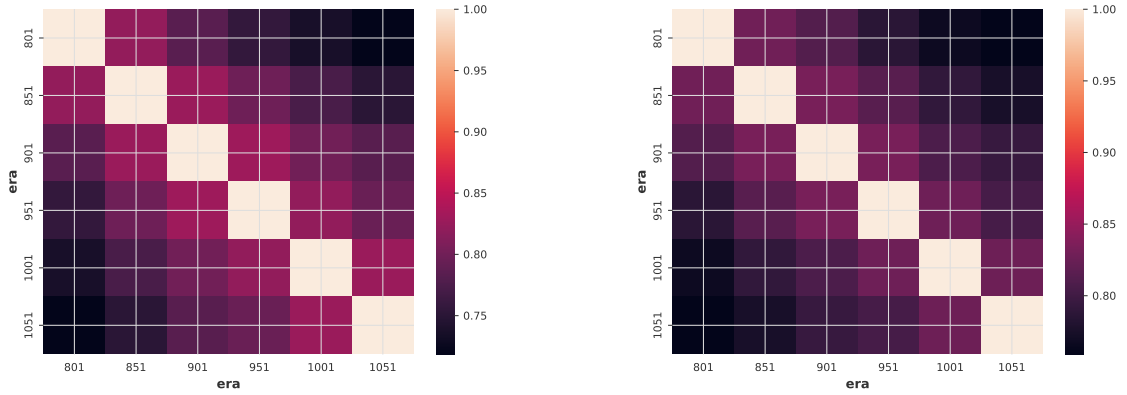
Figure 6: Performances of benchmark XGBoost models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$ for risk metrics (a) Mean Corr, (b) Sharpe ratio and (c) Calmar ratio under different market regimes

where $\textbf{Corr}(\cdot, \cdot)$ is the Pearson correlation function.

6.2.1 Differences between benchmark models

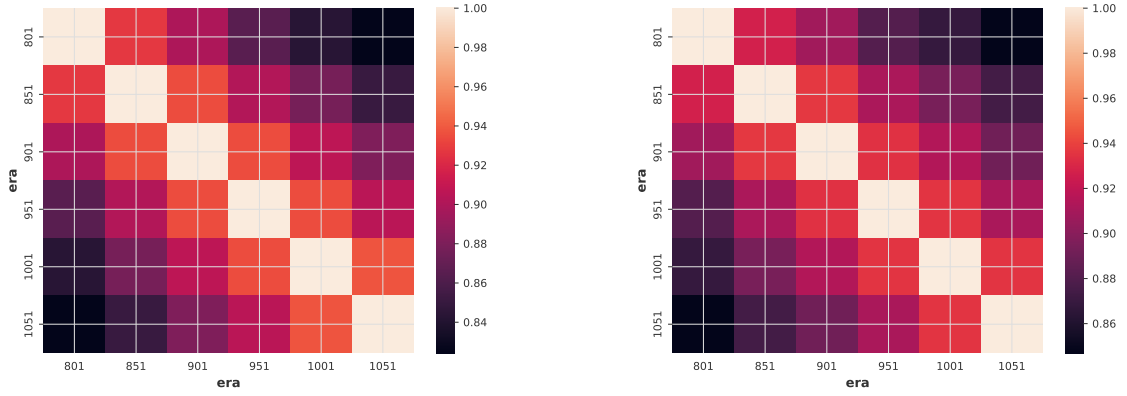
To understand the differences between models trained with different hyperparameters, we use the structural similarity measure in Section 7 to understand the overall structural similarity between the **Ansatz** and **Numerai** benchmark models.

In Figure 7, the temporal correlation structure of the benchmark models with sizes $B = 1000, 5000$ with the two hyperparameters sets are shown. Correlation between the retrained models decreases as the time gap between model retrains increases as expected. For smaller models ($B = 1000$), models are less correlated with each other. For larger



(a) Correlation between models with $B = 1000$ and Ansatz hyperparameters

(b) Correlation between models with $B = 1000$ and Numerai hyperparameters



(c) Correlation between models with $B = 5000$ and Ansatz hyperparameters

(d) Correlation between models with $B = 5000$ and Numerai hyperparameters

Figure 7: Temporal correlation structure of benchmark models from Eras 801 to Era 1051.

models ($B = 5000$), models are highly correlated with each other (with $\mathcal{S} > 0.9$) even after 150 eras. This is also why we do not retrain models with sizes $B \geq 10000$ as the models are expected to be highly correlated with each other within the validation and test period.

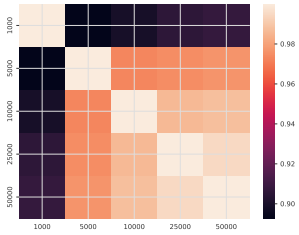
In Figure 8, correlation structure of benchmark models of different sizes at Era 801, the first model training time is shown. For both sets of hyperparameters, models with sizes $B \geq 5000$ are highly correlated. Cross correlation between models with different hyperparameters of the same sizes are slightly less correlated but the difference is negligible for models with sizes $B \geq 5000$.

From the above observation, we hypothesise that models with different tree structure hyperparameters converge to the theoretical learning limit when the number of boosting rounds B increases, on the condition that the learning rate L of model is selected by the Ansatz formula. This means hyperparameter optimisation is not necessary for large GBDT models. We can pick any reasonable hyperparameter sets, such as the Ansatz hyperparameter set, based on computational requirement.

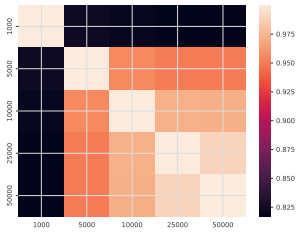
Combining with the fact model performances does not significantly improves beyond number of boosting rounds $B = 5000$, we conclude it is not necessary to train **single** models with size $B > 5000$ as it consumes more computational resources while not providing meaningful gain in model performances. It is better to allocate the computational resources to train an ensemble of models with size $B = 5000$ or less instead.

7 Deep IL XGBoost Models

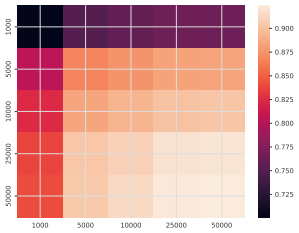
We now deploy the full deep IL model with dynamic ensembling, which models trained with different sampling schemes and hyperparameters are combined dynamically to create better models.



(a) Correlation between models with Ansatz hyperparameters



(b) Correlation between models with Numerai hyperparameters



(c) Cross Correlation between models with Ansatz and Numerai hyperparameters

Figure 8: Correlation structure of benchmark models of different sizes at Era 801

This is inspired by our previous work on dynamic forecasting in financial data [24] and in weather forecasting [56], where ensemble forecasting has been used to improve robustness of predictions. Instead of creating predictions based on a single set of data/parameters, multiple sets of data/parameters are used to capture a range of scenarios, which represent possible trajectories for the evolution of weather or financial systems.

We report performances of the deep IL models from Era 801 to Era 1070 by the following regimes. The validation period is Era 801 to Era 885. The test period is between Era 901 and Era 1070, where 15 eras of data embargo are used. We also report performance based on market regimes within the test defined to understand if models behave differently under different regimes.

- Validation: Era 801 - Era 885
- Test: Era 901 - Era 1070
- Bull: Era 901 - Era 1050
- Bear: Era 901 - Era 1070

A key assumption for model ensembling is to use a **diversified** set of base models that are not so correlated to achieve the variance reduction benefits during ensembling. As a result, we explore three different ways here in creating diversified base models using different sampling strategies. In particular, we study model ensembles created with different (1) feature sets, (2) learning rates and (3) training sizes.

Unless otherwise specified, we apply regular era sampling in training the XGBoost models in Layer 1, which in turn gives 4 different models for each deep IL ensemble strategy.

7.1 Ensemble strategies based on data sampling

7.1.1 Training Size Ensemble

For incremental learning problems, it is not known in advance how much data is required for model learning. Trade-offs are made when deciding the training sizes. If more data is used, the training data can cover more possible regimes that is seen historically but also have a risk including data no longer relevant. If less data is used, the training data can adapt to concept drift in data quicker but also increase the risk of overfitting the models towards the current data regime. Therefore, there is no universal rule to select the training set size.

The standard training set size recommended by Numerai is 600. Here, we explore if adjusting the training set sizes can improve model performances.

In Algorithm 4, the maximum training set size of Layer 1 models are increased to 800 eras and we train 5 models using the most recent $100\%, 87.5\%, 75\%, 62.5\%, 50\%$ of data. The number of boosting rounds is scaled with respect to training size. The learning rates of models are determined by the Ansatz formula $L = \frac{50}{B}$ using the scaled number of boosting rounds.

The Layer 2 models N_k, $1 \leq k \leq 2$ used in the above algorithm is defined as:

- N_1: Simple average over all predictions
- N_2: Ridge Regression with L2-regularisation $\alpha = 1e-4$ and parameters are restricted to be non-negative.

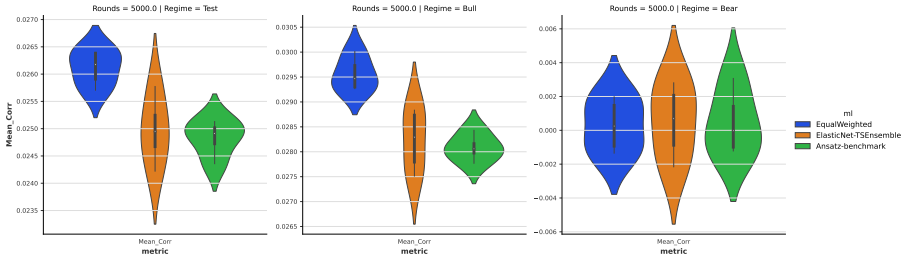
Algorithm 4: Deep IL XGBoost models over different training sizes

Input: Number of boosting rounds $B = 5000$, Max Training size of Layer 1 $X_1 = 800$, Data embargo $b_1 = 15$, $b_2 = 6$

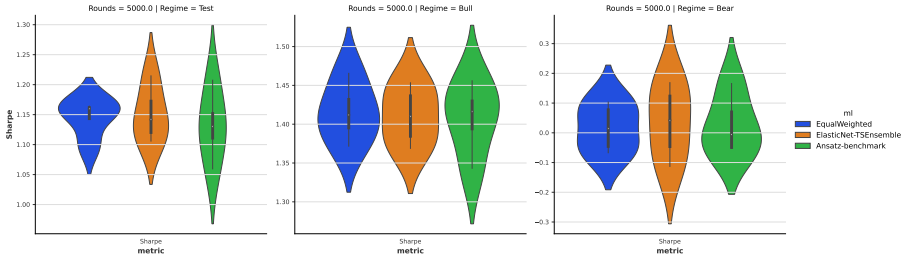
Set starting Era $D = 801$
Set Ansatz learning rate $L = \frac{50}{B}$
Set Lookback ratios $r_1 = 1.0$, $r_2 = 0.875$, $r_3 = 0.75$, $r_4 = 0.625$, $r_5 = 0.5$ **for** $1 \leq j \leq 5$ **do**
  Set Retrain period $T_j = \lfloor \frac{r_j X_1}{16} \rfloor$
  for $1 \leq i \leq \lfloor \frac{270}{T_j} \rfloor$ **do**
    Set $D_1 = D + (i-1)T$
    Set number of boosting rounds $B_j = r_j B$
    Set learning rate $l_j = \frac{L}{r_j}$
    Prepare training data $\mathcal{D}_j$ using r_j proportion of data from Era 1 to $D_1 - b_1 + (i-1)T$
    Train Layer 1 XGBoost models M_j^i with training data $\mathcal{D}_j$ using number of boosting rounds B_j with learning rate l_j, other hyperparameters are unchanged.
    Obtain model predictions for M_j^i from Era D_1 to Era $\min(D_1 + T_j, 1070)$
  end
end
for $1 \leq j \leq 170$ **do**
  Set $D_2 = D + 99 + j$
  for $1 \leq k \leq 2$ **do**
    Train Layer 2 models N_k using the Layer 1 model predictions from Era $D_2 - b_2 - 25$ to $D_2 - b_2$
    Obtain predictions from Layer 2 models N_k for Era $D_2 + 1$
  end
end

In Figure 9, we compare the performances of the two Layer 2 models (Elastic Net, Equal Weighted) with the benchmark Ansatz model of $B = 5000$. Equal Weighted model over all possible training set sizes achieves a higher Mean Corr than the benchmark model in the test period. The are improvement in the Bull market but not in the Bear market. Elastic Net model does not significantly improve the risk metrics compared to benchmark. Calmar ratio of Equal Weighted model is improved in the Bull market but not in the Bear market.

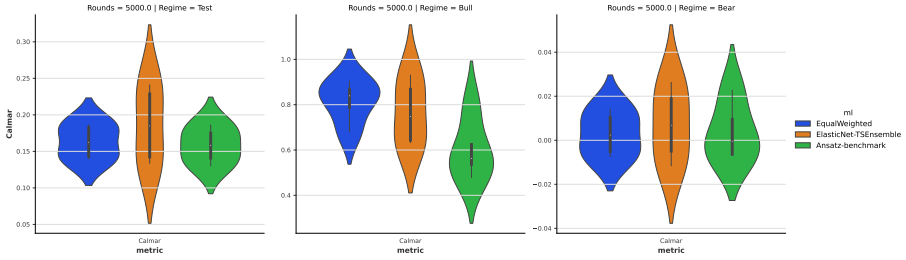
In Figure 10, XGBoost models with different training set sizes have structural similarity $0.7 < \mathcal{S} < 0.9$ at Era 801. While choosing a smaller training set size can decrease structural similarity, model performances are decreased at the same time, as shown in Figure 19 in SI. Therefore, model ensembling based on training set size is not efficient to create model ensemble, compared to the methods suggested below.



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 9: Performances, (a) Mean Corr, (b) Sharpe ratio and (c) Calmar ratio of the deep IL XGBoost models with different training sizes under different market regimes with $B = 5000$.

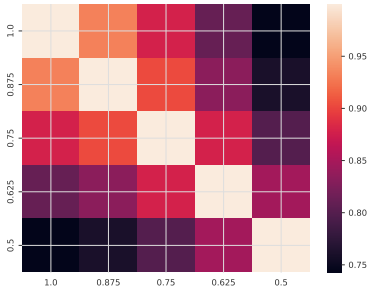


Figure 10: Structural similarity of models with different training set sizes at Era 801

7.2 Ensemble strategies based on different learning strategies

7.2.1 Learning Rate model ensemble (Complexity ensemble)

Recent research suggests features are learnt with different speeds within a neural network [57, 58]. Inspired by this idea, we combine GBDT models with different learning rates to learn models capturing both fast and slowing features.

In algorithm 5, we combine XGBoost models with 5 different learning rates. For a given number of boosting rounds B, in addition to training the model of size B as above, we train two larger models of size $2B$ and $4B$ and two smaller models of size $\frac{B}{2}$ and $\frac{B}{4}$ where the learning rate are adjusted by the Ansatz formula. To reduce computational costs, the

two larger models are not regularly retrained. Only models with number of boosting rounds less than or equal to B are regularly retrained. The Layer 2 models N_k used in algorithm 5 are the same as those used in algorithm 4.

As the Ansatz formula is used to determine the learning rate L and the number of boosting rounds B pair for the Layer 1 models, the above procedure is equivalent to combining models with different complexities, where the number of boosting rounds B is used to measure the complexity of GBDT models.

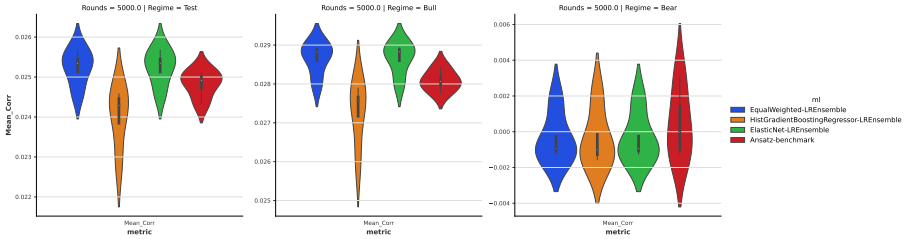
Algorithm 5: Deep IL XGBoost models over different learning rates

Input: Number of boosting rounds $B = 5000$, Training size of Layer 1 $X_1 = 585$, Retrain Frequency $T = 50$, Data embargo $b_1 = 15$, $b_2 = 6$

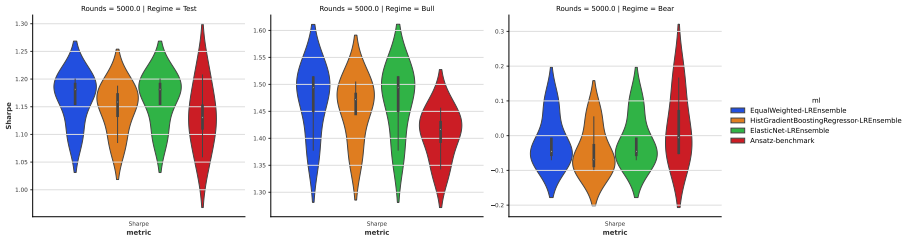
Set starting Era $D = 801$
Set Ansatz learning rate $L = \frac{50}{B}$
for $1 \leq i \leq 6$ **do**
  Set $D_1 = D + (i-1)T$
  Prepare training data from Era 201 to $D_1 - b_1 + (i-1)T$
  for $1 \leq j \leq 3$ **do**
    Train Layer 1 XGBoost model M_j^i, with number of boosting rounds rounds $B_j = \frac{2B}{2^j}$ and learning rate $L_j = \frac{2^j L}{2}$, other hyperparameters are unchanged.
    Obtain model predictions for M_j^i from Era D_1 to Era $\min(D_1 + 50, 1070)$
  end
end
Prepare training data from Era 201 to 800
for $4 \leq j \leq 5$ **do**
  Train Layer 1 XGBoost model M_j, with number of boosting rounds rounds $B_j = \frac{2^j B}{8}$ and learning rate $L_j = \frac{8L}{2^j}$, other hyperparameters are unchanged.
  Obtain model predictions for M_j from Era 801 to Era 1070
end
for $1 \leq j \leq 170$ **do**
  Set $D_2 = D + 99 + j$
  for $1 \leq k \leq 2$ **do**
    Train Layer 2 models N_k using the Layer 1 model predictions from Era $D_2 - b_2 - 25$ to $D_2 - b_2$
    Obtain predictions from Layer 2 models N_k for Era $D_2 + 1$
  end
end

We run Algorithm 5 for $B = 5000$ and performances for the two Layer 2 models are shown in Figure 11, compared with the **Ansatz** benchmark model with $B = 5000$. Equal Weighted and Elastic Net model can improve Mean Corr and Sharpe ratio in the test period compared to the benchmark. Calmar ratio is also improved in the Bull market but not Bear market.

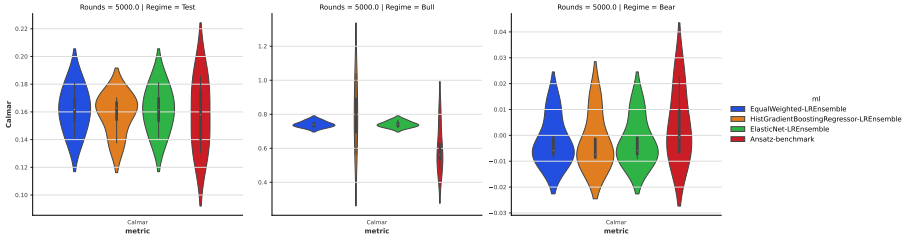
In Figure 20 in SI, we show the learning curves of the 5 Layer 1 XGBoost models with different learning rates and the corresponding number of boosting rounds (1250,2500,5000,10000,20000) are trained. Larger models performed slightly better than smaller models in the validation and test period. However, there are no significant differences in model performances in the Bear market. There is no single optimal complexity across all regimes and therefore using deep IL can combine the strength of models with different complexity (learning rates) so that the ensemble model are more robust.



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 11: Performances, (a) Mean Corr, (b) Sharpe ratio and (c) Calmar ratio of the deep IL XGBoost models with different learning rates under different market regimes.

7.3 Ensemble strategies based on different targets

Feature projection was used in Chapter **??** to reduce drawdown of trading strategies. In the V4.2 dataset [49], Numerai provides 5 different targets (Alpha-20D, Bravo-20D, Charlie-20D, Delta-20D, Echo-20D) in addition to the main scoring target (Cyrus-20D) which incorporates various risk management and hedging strategies. By design, these targets will offer a lower return (Mean Corr) but with a much lower risks (Max Drawdown and Volatility). The overall risk profile is improved even the portfolio return is reduced.

7.3.1 Model ensemble with different targets and learning rates

In algorithm 6, we combine XGBoost models trained with the five different targets using different learning rates as in algorithm 5. In total, we train 25 Layer 1 XGBoost models to be combined in Layer 2. The Layer 2 models N_k used in algorithm 6 are the same as those used in algorithm 4.

In Figure 12, we compare the performances of the two Layer 2 models (Elastic Net, Equal Weighted) with the benchmark Ansatz model of $B = 5000$. Equal Weighted model over all 25 Layer 1 XGBoost models with different targets over different learning rates achieves a higher Sharpe and Calmar ratio than the benchmark model in the test period at a lower Mean Corr ($\approx 90\%$ of the Benchmark model). Elastic Net can further improve the Calmar ratio but with a further lower Mean Corr ($\approx 75\%$ of the Benchmark model).

The improvement of Sharpe and Calmar of models using different targets can be attributed to a lower downside in the Bear market. Employing various hedging strategies, such as using the risk-controlled targets in model training will result in a lower performance in Bull market. The diversification benefits can only been observed when there are regime changes in the data, such as during the Bear market where the benchmark unhedged strategy perform poorly. Therefore, to fairly access the merit of different hedging strategies, the test period needs to be long enough to cover different market regimes.

Algorithm 6: Deep IL XGBoost models over different targets using different learning rates

Input: Number of boosting rounds $B = 5000$, Training size of Layer 1 $X_1 = 585$, Retrain Frequency $T = 50$, Data embargo $b_1 = 15$, $b_2 = 6$

Set starting Era $D = 801$
Set Ansatz learning rate $L = \frac{50}{B}$
Set Learning Targets y_1, y_2, y_3, y_4, y_5 to be Alpha-20D, Bravo-20D, Charlie-20D, Delta-20D, Echo-20D
for $1 \leq k \leq 5$ **do**
  for $1 \leq i \leq 6$ **do**
    Set $D_1 = D + (i-1)T$
    Prepare training data from Era 201 to $D_1 - b_1 + (i-1)T$
    for $1 \leq j \leq 3$ **do**
      Train Layer 1 XGBoost model $M^i_{j,k}$, with number of boosting rounds rounds $B_j = \frac{2B}{2^j}$ and learning rate $L_j = \frac{2^j L}{2}$ using target y_k , other hyperparameters are unchanged.
      Obtain model predictions for M^i_j from Era D_1 to Era $\min(D_1 + 50, 1070)$
    end
  end
end
Prepare training data from Era 201 to 800
for $1 \leq k \leq 5$ **do**
  for $4 \leq j \leq 5$ **do**
    Train Layer 1 XGBoost model $M_{j,k}$, with number of boosting rounds rounds $B_j = \frac{2^j B}{8}$ and learning rate $L_j = \frac{8L}{2^j}$ using target y_k, other hyperparameters are unchanged.
    Obtain model predictions for M_j from Era 801 to Era 1070
  end
end
for $1 \leq j \leq 170$ **do**
  Set $D_2 = D + 99 + j$
  for $1 \leq k \leq 2$ **do**
    Train Layer 2 models N_k using the Layer 1 model predictions from Era $D_2 - b_2 - 25$ to $D_2 - b_2$
    Obtain predictions from Layer 2 models N_k for Era $D_2 + 1$
  end
end

7.4 Ensemble strategies based on feature sampling

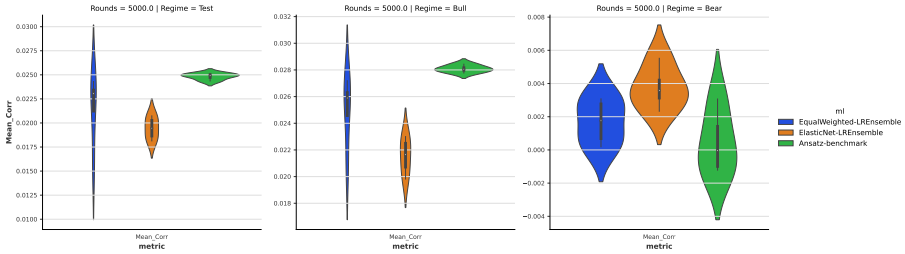
7.4.1 Feature Sets model ensemble

Feature selection and sampling methods are useful in training models. (Random) feature sampling, a common procedure used at the **local** level before the start of training each tree can be applied at the **global** level before model training. By design, the learnt models are more diverse. It also lowers computational requirements of models as we do not have to fit all the data to the model. Here, we explore if using Jackknife sampling [59] or other sampling techniques can build diversified models suitable for ensembling.

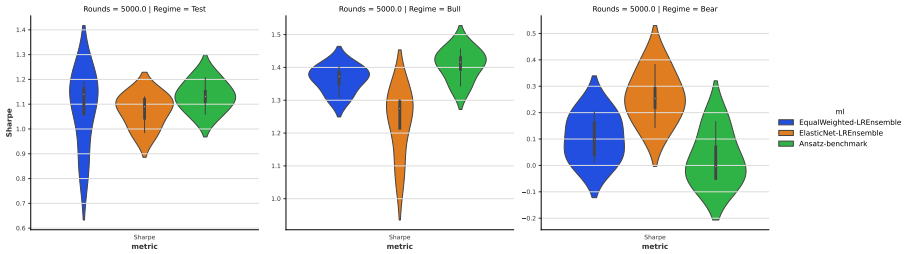
The feature group labels provided by Numerai in V4.2 dataset can be considered as a way of feature clustering using domain knowledge. Instead of analysing the high-dimensional temporal correlation structure of the features by clustering or dimensionality reduction methods, we use the labels provided by Numerai, which is built with the knowledge of data sources and feature generation process to correctly group features into different categories. This approach can save computational time and avoids identifying spurious relationships between features.

Jackknife feature sets $\mathcal{F}_j$, for $1 \leq j \leq 10$ are created as follows. For each for the 10 feature groups (Intelligence, Charisma, Strength, Dexterity, Constitution, Wisdom, Agility, Serenity, Sunshine, Rain), we remove that set from the 2132 features one at a time, and then use to remaining 9 features groups to form the Jackknife feature sets $\mathcal{F}_1 \ldots \mathcal{F}_{10}$. The Jackknife feature sets are then used to train deep IL XGBoost models, using the procedure described in algorithm 7. The Layer 2 models N_k used in algorithm 7 are the same as those used in algorithm 4.

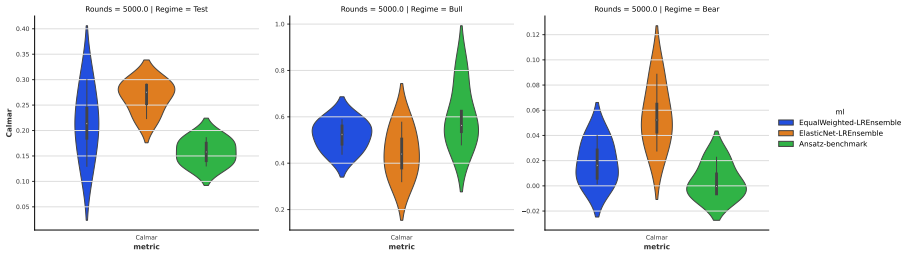
To evaluate the usefulness of feature group labels in model building, we compare our approach with two different baseline methods, (i) Deep IL XGBoost models over random feature sampling, which the procedure is described in



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 12: Performances, (a) Mean Corr, (b) Sharpe ratio and (c) Calmar ratio of the deep IL XGBoost models with different targets and learning rates under different market regimes.

algorithm 8. and (ii) benchmark XGBoost models trained with all the features using **Ansatz** hyperparameters described as above. The Layer 2 models N_k used in algorithm 8 are the same as those used in algorithm 7.

The reason to use (i) as benchmark is to see if feature group labels offers information that is better than random in separating the features into groups representing different signal sources and thus create information barrier between models such that models will be forced to learn rules that are different from each other, and thus reduce correlation between predictions. The reason to use (ii) as benchmark is to check if any form of feature selection is beneficial to model performances at all.

We run the algorithms for $B = 5000$. In Figure 14 we show the performances of the four Layer 2 models from Jackknife and random feature sampling with the benchmark **Ansatz** model with $B = 5000$, which are also regularly retrained.

The Layer 2 models from Jackknife sampling have a higher Mean Corr and Sharpe ratio than the models from random sampling and the benchmark model in both the validation and test period. The Layer 2 models using random sampling has comparable Mean Corr and Sharpe ratio with the benchmark model in both the validation and test period. Within models using Jackknife sampling, there are no significant difference between the Equal Weighted and Elastic Net model. However, for models using random sampling, Elastic Net model underperformed Equal weighted model. The historical performances of models from random sampling are simply noise and we are not supposed to be able to learn any useful patterns from that.

In Figure 13, we compare the correlation between the 10 Layer 1 XGBoost models obtained by Jackknife and random feature sampling. Models trained **without** rain feature set are uncorrelated to the rest of the models. Models trained by removing the other nine feature sets one at a time have correlation lower than 0.86 with average structural similarity of 0.66. Structural similarity of models obtained by Jackknife sampling is also stable across time, demonstrated by the similar heatmap representation of the models structural similarity at Era 801,901,1001. Models obtained by random feature sampling do not have any stable structural similarity by design.

Algorithm 7: Deep IL XGBoost models over feature set Jackknife sampling

Input: Number of boosting rounds $B = 5000$, Training size of Layer 1 $X_1 = 585$, Retrain Frequency $T = 50$, Data embargo $b_1 = 15$, $b_2 = 6$

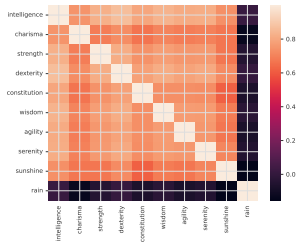
Set starting Era $D = 801$
Set Ansatz learning rate $L = \frac{50}{B}$
for $1 \leq i \leq 6$ **do**
  Set $D_1 = D + (i-1)T$
  Prepare training data from Era 201 to $D_1 - b_1 + (i-1)T$
  for $1 \leq j \leq 10$ **do**
    Train Layer 1 XGBoost models M_j^i, with feature set $\mathcal{F}_j$, other hyperparameters are unchanged.
    Obtain model predictions for M_j^i from Era D_1 to Era $\min(D_1 + 50, 1070)$
  end
end
for $1 \leq j \leq 170$ **do**
  Set $D_2 = D + 99 + j$
  for $1 \leq k \leq 2$ **do**
    Train Layer 2 models N_k using the Layer 1 model predictions from Era $D_2 - b_2 - 25$ to $D_2 - b_2$
    Obtain predictions from Layer 2 models N_k for Era $D_2 + 1$
  end
end

Algorithm 8: Deep IL XGBoost models over random feature sampling

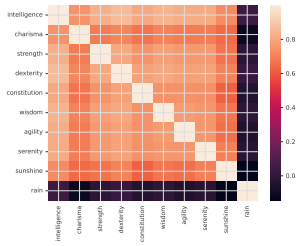
Input: Number of boosting rounds B, Training size of Layer 1 $X_1 = 585$, Retrain Frequency $T = 50$, Data embargo $b_1 = 15$, $b_2 = 6$

Set starting Era $D = 801$
Set Ansatz learning rate $L = \frac{50}{B}$
for $1 \leq i \leq 6$ **do**
  Set $D_1 = D + (i-1)T$
  Prepare training data from Era 201 to $D_1 - b_1 + (i-1)T$
  for $1 \leq j \leq 10$ **do**
    Train Layer 1 XGBoost models M_j^i, with 50% of the 2132 features selected by random without replacement, other hyperparameters are unchanged.
    Obtain predictions for M_j^i from Era D_1 to Era $\min(D_1 + 50, 1070)$
  end
end
for $1 \leq j \leq 170$ **do**
  Set $D_2 = D + 99 + j$
  for $1 \leq k \leq 2$ **do**
    Train Layer 2 models N_k using the Layer 1 model predictions from Era $D_2 - b_2 - 25$ to $D_2 - b_2$
    Obtain predictions from Layer 2 models N_k for Era $D_2 + 1$
  end
end

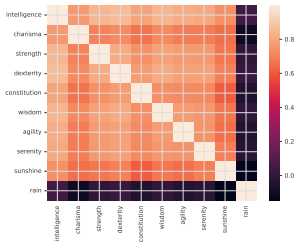
While highly similar models offer limited diversification benefits in model ensembling, uncorrelated models created by random failed to generate better predictions. Using domain knowledge about the data generation process, we can create models that are not over-similar to each other which can significantly improve model performances after ensembling.



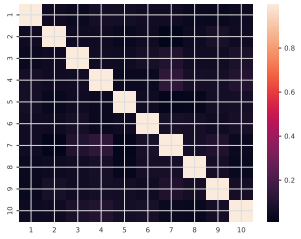
(a) Structural similarity of models with Jackknife feature sampling at Era 801



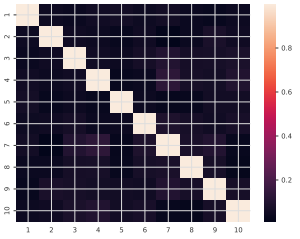
(b) Structural similarity of models with Jackknife feature sampling at Era 901



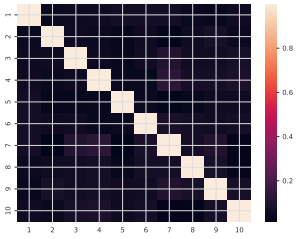
(c) Structural similarity of models with Jackknife feature sampling at Era 1001



(d) Structural similarity of models with random feature sampling at Era 801



(e) Structural similarity of models with random feature sampling at Era 901



(f) Structural similarity of models with random feature sampling at Era 1001

Figure 13: Structural similarity of models with Jackknife and random feature sampling at Era 801, 901, 1001

7.5 Dynamic Hedging based on model variances

Recent research suggests disagreement between investors are indicative signals of stock returns [**bali2023machine**]. In particular, under the Bull market, stocks that have the highest degree of disagreement between investors will under-perform stocks that have the lowest degree of disagreement between investors. The opposite holds under the Bear market.

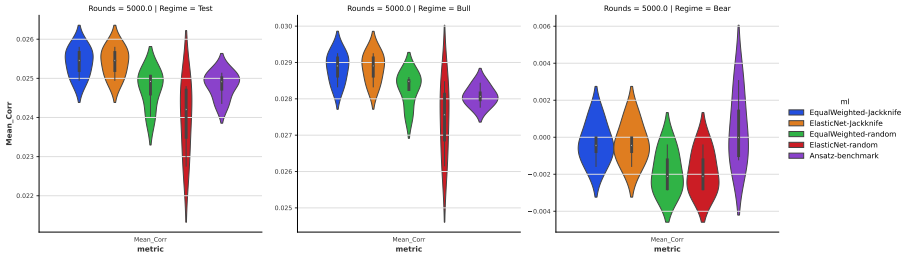
Here, the variance between model predictions from different Layer 1 models are used as proxy of disagreement between investors. In Algorithm 9, two strategies are built based on predictions from the Layer 1 models, namely the **Baseline** model predictions based on the simple average and the **Tail Risk** model predictions based on the standard deviation. The tail risk model will buy stocks that the investors (Layer 1 models) disagree with each other the most and sell stocks that the investors agree with each other the most. Two approaches are the used to combine the **Baseline** and **Tail Risk** model predictions. With **Static** hedging, a linear combination of 60% **Baseline** and 40% **Tail Risk** is used for the whole test period. With **Dynamic** hedging, the hedging ratio, which determines how much **Tail Risk** strategy is used are adjusted according to the prevailing performances of the **Tail Risk** strategy. The hedging ratio is switched between two modes, (i) No hedging and (ii) 40% **Baseline** and 60% **Tail Risk** according to the most recent 50 week performances of the **Tail Risk** strategy.

In Table 2, we compare the dynamic hedge model from deep IL XGBoost ensemble using feature set Jackknife, using the dynamic hedging strategy described above, with the example model provided by Numerai. The example model are trained using 100% of data which is not replicated directly here due to memory limits. We used regular era sampling to train 4 models each with 25% of data and then take the simple average over those.

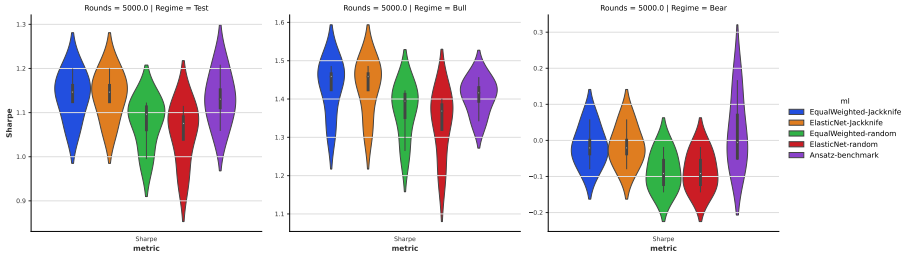
The **Tail Risk** model works well when the **Baseline** model has poor performances, demonstrating the complementary nature. The **Dynamic** hedged model achieves comparable Mean Corr with the example model provided by Numerai. The Sharpe ratio are improved from 0.9626 to 1.3169 while the Max Drawdown reduces from 0.2608 to 0.0237, a more than 90% reduction. The portfolio return curve are much smoother for the **Dynamic** hedged model, as shown in Figure 16.

We repeat the dynamic hedging procedure on deep IL XGBoost models with random feature sampling, over different targets, learning rates and training sizes. Detailed results are shown in Tables 6,9,8,7 in SI. The dynamic hedged models from these ensembles have a lower Mean Corr and Sharpe ratio than the above.

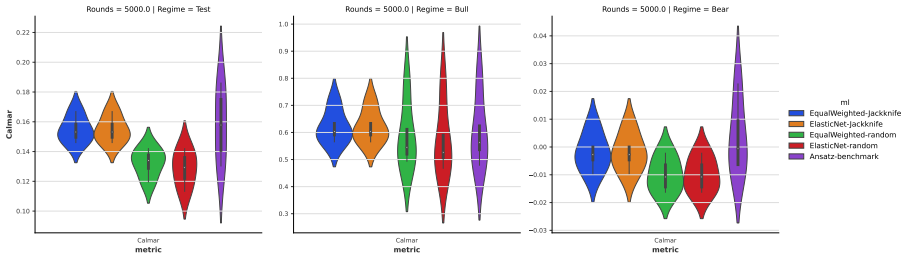
Model ensembles created based on Jackknife feature set sampling uses knowledge about the dataset and thus offer a better approximation of disagreement between investors. Therefore, Tail Risk model created based on the variance between models trained with Jackknife feature set sampling are the most effective hedging strategy in the Bear market.



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 14: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with Jackknife and random feature sampling under different market regimes.

8 Conclusion

In this study, both traditional tabular and factor-timing models are studied for the IL problem on the temporal tabular dataset from Numerai. Traditional tabular models, if retrained regularly can adapt to distribution shifts in data. On the other hand, factor-timing models failed to adapt to distribution shifts in data.

GBDT are robust ML models We found that GBDT models is the best machine learning method for the Numerai datasets, agreeing with the findings of [37], which demonstrate the robust and superior performances of GBDT models on large datasets. This is also partly due to the nature of features, being binned values from continuous underlying measures, which favours models based on decision rules rather than regression. With suitable designs of the training process, such as a slow learning rate with a large number of boosting rounds, we can train XGBoost models with good performances, slowly converging to the theoretical optimal.

GBDT models are highly scalable and have robust performance over slightly perturbed hyperparameters. The larger the GBDT model, the lesser the effect of model hyperparameters on the learning process and model performances. GBDT models are more numerically stable as it does not have convergence issues which is common in training neural networks. For example, vanishing gradient is a common issue in training MLPs therefore different variations of activation functions are proposed.

Feature and Data Sampling Data management and forgetting mechanism is an integrated part of an IL pipeline [3] to build robust prediction models on a data stream. The impact of data sampling methods are usually over-looked in most quantitative finance research and even in hedge funds [25]. Data and feature sampling methods can have significant effects on model performances.

Algorithm 9: Model Disagreement

Input: At era t: predicted values $\hat{y}_k^t \in \mathbb{R}^{N_t}$ from Layer 1 models $\mathcal{M}_k$, $1 \leq k \leq K$, and temporal tabular dataset $X_t \in \mathbb{R}^{N_t \times M}$ where M is the number of features

Output: Hedged model predictions $\hat{h}^t \in \mathbb{R}^{N_t}$

Calculate normalised predictions $\hat{r}_t$ from Layer 1 models

$$\hat{r}_t = \text{rank}(\hat{y}_k^t) - 0.5,$$

where the rank function calculates the percentile rank of a value within a vector, so that $-0.5 \leq \hat{r}_t \leq 0.5$.

Calculate Baseline model predictions (average) $\hat{y}_{mean}^t = \text{rank}\left(\frac{1}{K}\sum_{k=1}^{K}\hat{y}_k^t\right) - 0.5$

Calculate Tail risk model predictions (standard deviation) $\hat{y}_{sd}^t = \text{rank}\left(\sqrt{\frac{1}{K}\sum_{k=1}^{K}(\hat{y}_k^t - \hat{y}_{mean}^t)^2}\right) - 0.5$

Calculate static hedged model predictions $\hat{h}_t = 0.6\hat{y}_{mean}^t + 0.4\hat{y}_{sd}^t$

Calculate the average recent performance of tail risk model $\bar{\rho}_t$, $\bar{\rho}_t = \frac{1}{50}\sum_{i=t-56}^{t-7}\rho_i$, where ρ_i is calculated by the scoring formula in Section **??**.

if $\bar{\rho}_t \geq 0$ **then**
  Set Hedge ratio $h = 0.6$
end
else
  Set Hedge ratio $h = 0$
end

Calculate dynamic hedged model predictions $\hat{d}_t = (1-h)\hat{y}_{mean}^t + h\hat{y}_{sd}^t$

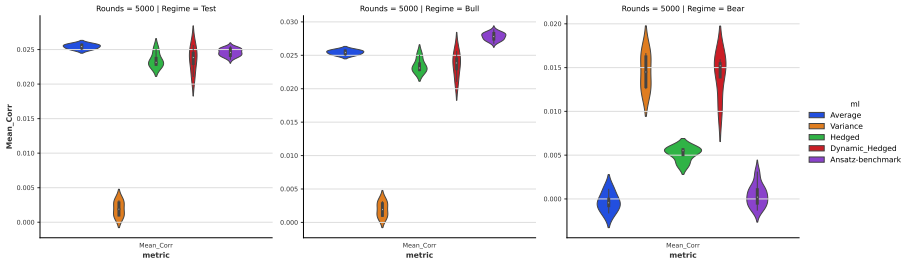
Regime	Strategy	Mean Corr	Sharpe	Max Drawdown
Test	Example Model	0.0264	0.9626	0.2608
	Baseline Model	0.0266	1.1725	0.1602
	Tail Risk Model	0.0023	0.1601	0.2385
	Static Hedged Model	0.0203	1.2159	0.0435
	Dynamic Hedged Model	**0.0266**	**1.3169**	**0.0237**
Bull	Example Model	**0.0307**	1.2512	0.0693
	Baseline Model	0.0302	**1.4780**	0.0396
	Tail Risk Model	0.0006	0.0435	0.2385
	Static Hedged Model	0.0215	1.3014	0.0309
	Dynamic Hedged Model	0.0283	1.3844	**0.0237**
Bear	Example Model	-0.0060	-0.2306	0.2608
	Baseline Model	-0.0002	-0.0080	0.1602
	Tail Risk Model	**0.0153**	**1.2929**	**0.0000**
	Static Hedged Model	0.0109	0.7489	0.0435
	Dynamic Hedged Model	0.0137	1.1500	0.0220

Table 2: Performances of Dynamic Hedged deep IL XGBoost ensemble model based on feature set Jackknife sampling and V4.2 Example Model from Era 901 to Era 1070 under different market regimes.

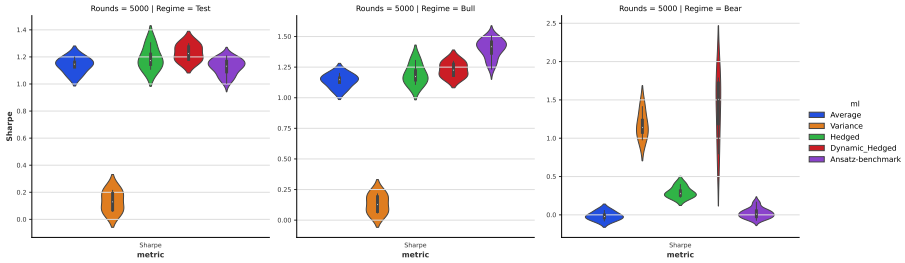
Removing data with targets equal to the Median value can reduce computational time by half without significant loss to model performances. This demonstrate well designed sampling procedures can be used to filter data for effective model training.

Feature sampling can be used to increase diversity of models by enforcing constraints on features that are allowed to be used or interact in a model. Using feature set group labels can create feature sets that are more efficient in creating diverse model ensembles than random sampling.

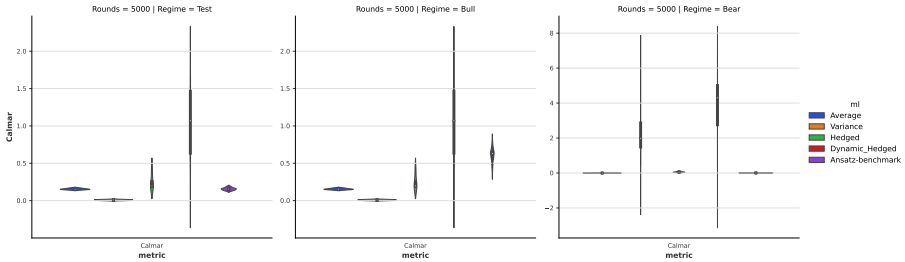
In all incremental problems we encounter the stability-plasticity dilemma [60], which is the trade-off between the ability of ML models to adapt to new patterns and preserve existing knowledge. It is not known in advance which data sampling method will have the optimal performance and therefore ensembling models with different training sizes with equal weights is often a robust strategy when there are no additional information to decide how much data to use for model training.



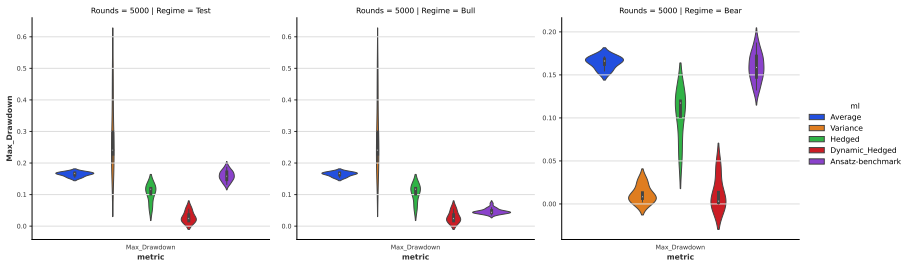
(a) Mean Corr



(b) Sharpe



(c) Calmar



(d) Max Drawdown

Figure 15: Performances, (a) Mean Corr, (b) Sharpe ratio, (c) Calmar ratio and (d) Max Drawdown of the deep IL XGBoost models with Jackknife feature sampling and dynamic hedging under different market regimes.

Retraining benchmark models regularly can improve performances significantly compared to using the same model without updating. In general, model performances improves with the frequency of retrain but the requirements on computational resources also increase. Therefore, trade-offs between computational costs and the marginal gain in model performances are made for practical IL systems.

Learning rates and model complexity We derive an Ansatz formula to determine the learning rate L for a GBDT model given a fixed number of boosting rounds B. We show the formula is optimal for our benchmark GBDT models over a wide range of sizes, from $B = 1000$ to $B = 50000$.

Combining models with different degree of complexity, created with different number of boosting rounds with learning rates derived using the Ansatz formula can improve model performances compared to models using a fixed number

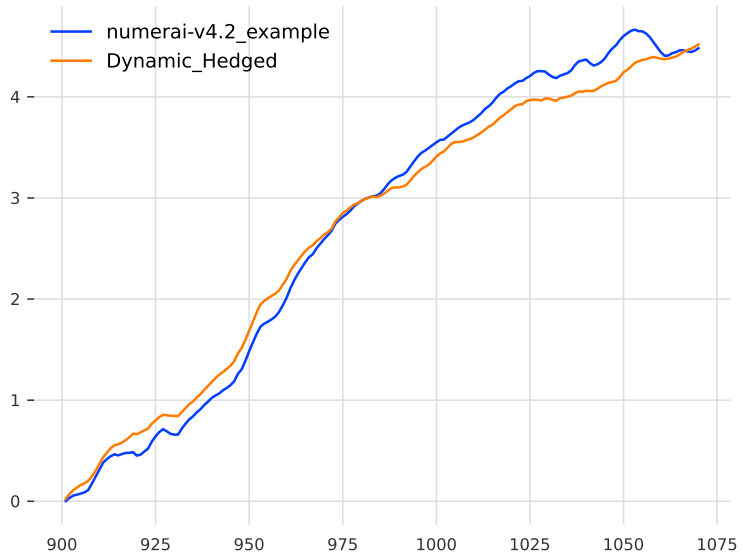


Figure 16: The portfolio return curve of the Dynamic Hedge model based on feature set Jackknife sampling and V4.2 Example Model from Era 901 to Era 1070

of boosting rounds. The optimal model complexity is regime dependent and therefore using deep IL techniques to dynamically combine model predictions can reduce downside risks in models.

Connection with Model Stacking/Selection Stacking is a simple but highly effective technique to combine different ML model predictions. The concept of stacking is not limited to machine learning. In finance, portfolio optimisation are studied in detail to improve investment returns, where a convex optimisation is solved at each time step to find the linear combination of assets or strategies that maximise risk-adjusted return.

Under the IL framework, model stacking can be performed dynamically. Here, we combine the predicted rankings from different ML models at each era with different weights. Instead of considering model stacking as a *separate* step to model training, model stacking can be incorporated as an integrated part in the IL framework, as an extra layer in the IL model.

Hedging against regime changes Using the variance between models within the ensemble as a signal, tail risk strategy can be created to hedge the baseline prediction strategy based on simple average of different component models. Regime changes in data can be captured by uncertainty of model predictions, as a higher variance between models within the ensemble suggests a lower confidence of the predictions. Therefore, prediction based on variance would perform well under regime changes, such as the Bear market period identified in this study. The best performance is achieved when the component models are trained using different combinations of feature subsets based on economic knowledge about the dataset. In this case, the variance between models are the most informative approximation of disagreement between investors in the stock market.

Further Work In most practical applications, *multiple* machine learning methods are used together to create an ensemble prediction. The IL model presented in this paper provides a comprehensive way to integrate different ML models in a consistent and systematic way to create point-in-time predictions. With a multi-layer structure and modularised design within each layer, the deep IL model can flexibly model datasets with different complexities and structures. Further work can be done by integrating different deep tabular models into the model and bench-marking different machine learning methods under the IL framework.

Within our incremental learning framework, we retrain each XGBoost model from scratch without using any information from previous ones. Currently, new methods [61, 62] have been developed which adapt towards concept drift in data by adding a suitable amount of trees to existing GBDT models. Different approaches, such as reusing a certain amount of base learners (trees) from previous trained GBDT models or updating the weights of trees dynamically depending on the severity of concept drift can be explored in future work.

We only consider the most simplest form of deep learning models, MLP in this paper. Recent research suggests regularisation techniques [40] can improve performances of neural networks models over a wide range of network architecture. Further can be done to investigate if careful design of the model training process with suitable regularisation can improve the scalability and model performances.

9 Supplementary Information

9.1 Algorithms of different benchmark machine learning models studied

9.1.1 Signature Transforms

Signature transforms are applied on continuous paths. A path X is defined as a continuous function from a finite interval $[a, b]$ to $\mathbb{R}^d$ with d the dimension of the path. X can be parameterised in coordinate form as $X_t = (X_t^1, X_t^2, \dots, X_t^d)$ with each X_t^i being a single dimensional path.

For each index $1 \leq i \leq d$, the increment of i-th coordinate of path at time $t \in [a, b]$, $S(X)_{a,t}^i$, is defined as

$$S(X)_{a,t}^i = \int_{a<s<t} \mathrm{d}X_s^i = X_t^i - X_a^i$$

As $S(X)_{a,\cdot}^i$ is also a real-valued path, the integrals can be calculated iteratively. A k-fold iterated integral of X along the indices $i_1, \dots, i_k$ is defined as

$$S(X)_{a,t}^{i_1,\dots,i_k} = \int_{a<t_k<t} \cdots \int_{a<t_1<t_2} \mathrm{d}X_{t_1}^{i_1} \dots \mathrm{d}X_{t_k}^{i_k}$$

The Signature of a path $X : [a, b] \mapsto \mathbb{R}^d$, denoted by $S(X)_{a,b}$, is defined as the infinite series of all iterated integrals of X, which can be represented as follows

$$\begin{aligned} S(X)_{a,b} &= (1, S(X)_{a,b}^1, \dots, S(X)_{a,b}^d, S(X)_{a,b}^{1,1}, \dots) \\ &= \bigoplus_{n=1}^{\infty} S(X)_{a,b}^n \end{aligned}$$

An alternative definition of signature as the response of an exponential nonlinear system is given in [33].

Log Signature can be computed by taking the logarithm on the formal power series of Signature. No information is lost as it is possible to recover the (original) Signature from Log Signature by taking the exponential [32, 33]. Log Signature provides a more compact representation of the time series than Signature.

$$logS(X)_{a,b} = \bigoplus_{n=1}^{\infty} \frac{(-1)^{(n-1)}}{n} S(X)_{a,b}^{\otimes n}$$

Signatures can be computed efficiently using the Python package signatory [63]. The signature is a multiplicative functional in which Chen's identity holds. This allows quick computation of signatures on overlapping slices in a path. Signatures provide a unique representation of a path which is invariant under reparameterisation [32, 33]. Rough Path Theory suggests the signature of a path is a good candidate set of linear functionals which captures the aspects of the data necessary for forecasting. In particular, continuous functions of paths are approximately linear on signatures [64]. This can be considered as a version of the universal approximation theorem [42] for signature transforms.

Limitations for signature transforms in high dimensional datasets The number of signatures and log-signatures increases exponentially with the number of channels. For time series with a large number of channels, random sampling can be applied to select a small number ($5 < N < 20$) of time series with replacement from the original time series on which signature transforms are applied. Random sampling can be repeated a given number of times to generate representative features of the whole multivariate time series. Similar ideas are considered in [65], in which random projections on the high dimensional time series are used to reduce dimensionality before applying signature transforms.

Let $\tilde{X}$ be a multivariate time series with T time-steps and d dimensional features, denote $\tilde{X}_s \in \mathbb{R}^d$ be the observation of the time series at timestep s. Procedure 10 can be used to obtain paths, which are slices of time series with different lookback windows. Random Signature transforms 11 can then be used to compute the signature of the path, which summarises the information of the time series.

9.1.2 Random Fourier Transforms

Random Fourier Transforms are used in [34] to model the return of financial price time series. They can be applied to the feature performance time series at each time step as in Algorithm 12. The key idea is to approximate a mixture model of Gaussian kernels with trigonometric functions [35].

Algorithm 10: Lookback Window Slicing

Input: time series $\tilde{X} \in \mathbb{R}^{T \times d}$, lookback δ
Output: paths $X_t \in \mathbb{R}^{t \times d}$
for $1 \leq t \leq T$ **do**
  Set start of slice $s_1 = \max(1, t - \delta)$;
  Set end of slice $s_2 = t$;
  $X_t = (\tilde{X}_{s_1}, \tilde{X}_{s_1+1}, \ldots, \tilde{X}_{s_2})$;
end

Algorithm 11: Random Signature Transform

Input: path $X_t \in \mathbb{R}^{t \times d}$, level of signature L, number of channels C, number of feature sets p, where $d > C$;
Output: log signatures $s_t \in \mathbb{R}^{pN}$
Define $N =$ Number of Log Signatures of a path with C channels up to level L ;
for $1 \leq i \leq p$ **do**
  Sample with replacement C Columns from X_t, defined as $\tilde{X}_t^i$;
  Compute the Log Signatures $s_t^i \in \mathbb{R}^N$ of $\tilde{X}_t^i$;
end
Combine all log signatures $s_t = (s_t^1, \ldots, s_t^p)$

Algorithm 12: Random Fourier Transform [34]

Input: signal vector $x_t \in \mathbb{R}^d$, number of features sets p,
Output: transformed vector $s_t \in \mathbb{R}^{14p}$
for $1 \leq i \leq p$ **do**
  Sample $w_i \sim \mathcal{N}(0, I_{d \times d})$;
  Set grid $(\gamma_i)_{i=1}^1 4 = (0.1, 0.5, 1, 2, 4, 8, 16, 0.1, 0.5, 1, 2, 4, 8, 16)$;
  for $1 \leq j \leq 7$ **do**
    Set $s_{t,14i+j} = \frac{1}{\sqrt{7p}} \sin(\gamma_j w_i^T x_t)$
  end
  for $8 \leq j \leq 14$ **do**
    Set $s_{t,14i+j} = \frac{1}{\sqrt{7p}} \cos(\gamma_j w_i^T x_t)$
  end
end

9.1.3 Gradient Boosting Models

XGBoost Implementation XGBoost [66] modifies the above "standard" gradient boosting algorithms with approximation algorithms in split finding. Instead of finding the best(exact) split by searching over all possible split points on all the features, a histogram is constructed where splitting is based on percentiles of features. XGBoost supports two different growth policies for the leaf nodes, where nodes closest to the root are split (depth-wise) or the nodes with the highest change of loss function are split (loss-guide). The default tree-growing policy is depth-wise and performs better in most benchmark studies. XGBoost also supports L1 and L2 regularisation of model weights. Other standard model regularisation techniques such as limiting the maximum depth of trees and the minimum number of data samples in a leaf node are also supported.

Model Snapshots For GBDT models, it is easy to extract model snapshots, defined as the model parameters captured at the different parts of the training process. This can be done without any additional memory costs at inference.

Model snapshots of a GBDT model can be obtained as follows. The snapshots start with the first tree and the number of trees to be used is set to be $10\%, 20\%, \ldots, 100\%$ of the number of boosting rounds. This trivially gives 10 different GBDT models representing different model complexities from a *single* model.

9.1.4 Deep Learning Models

Training process PyTorch Lightning [67] is used to build neural network models as it supports modular design and allows rapid prototyping. Early stopping is applied based on the validation set based on a given number of rounds (patience). The batch size of the neural network is set to be the size of each era. The Adam optimiser in PyTorch with the default settings for the learning rate schedule is used. L2-regularisation on the model weights is also applied. Gradient clipping is also be applied to prevent the gradient explosion problem for correlation-based loss functions.

Architecture The network architecture is a sequential neural network with two parts, firstly a "Feature Engineering" part which consists of multiple feature engineering blocks and then the "funnel" part which is a standard MLP with decreasing layer sizes.

Each feature engineering block has an Auto-Encoder-like structure, where the number of features is unchanged after passing each block. Setting a neuron scale ratio of less than 1 corresponds to the case of introducing a bottleneck to the network architecture so as to learn a latent representation of data in a lower dimensional space. Algorithm 13 shows how to create the feature engineering part of the network.

Funnel architecture, as used in [68] is an effective way to define the neuron sizes in a network for different input feature sizes. Algorithm 14 shows how to create the funnel part of the network.

Each Linear layer is followed by a ReLU activation layer and dropout layer where 10% of weights are randomly zeroed.

Definition 1 (Linear Layer)**.**
A Linear Layer (M_1, M_2) within a sequential neural network is a transformation $X_2 = f(X_1)$ with input tensor $X_1 \in \mathbb{R}^{N \times M_1}$ and output tensor $X_2 \in \mathbb{R}^{N \times M_2}$ where N is the batch size of data. For a given non-linear activation function $\sigma(\cdot)$ such as ReLU, let $W \in \mathbb{R}^{M_2 \times M_1}$ be the weight tensor and $b \in \mathbb{R}^{M_2}$ be the bias tensor to be learnt in the training process, the Linear layer is defined as

$$f(X_1) = \sigma(X_1 W^T + b)$$

Algorithm 13: Feature Engineering network architecture

Input: Input feature size M, Number of encoding layers L, neuron scale ratio r
Output: Sequential Feature Engineering Network Architecture
for $1 \leq l \leq L$ **do**
  Encoding Layer l: Linear layer $(M, M * r)$
  Decoding Layer l: Linear layer $(M * r, M)$
end

Algorithm 14: Funnel network architecture

Input: Input feature size M, Output feature size K, Number of intermediate layers L, neuron scale ratio r
Output: Sequential Funnel Network Architecture
Input Layer: Linear layer (M, $M * r$)
for $1 \leq l \leq L$ **do**
  Intermediate Layer l: Linear layer $(M * r^l, M * r^{l+1})$
end
Output Layer: Linear layer $(M * r^{L+1}, K)$

Feature projection and Loss Function Pearson correlation calculated on the whole *era* of target and predictions is used as the loss function at each training epoch. Feature projection, if needed, can be applied from the outputs of network architecture. The neutralised predictions are further standardised to zero mean and unit norm. The negative Pearson correlation of the standardised predictions and targets is then used as the loss function to train the network parameters.

9.2 Creating benchmark XGBoost models

Example models provided by Numerai are trained under different conditions with the models we presented here. In particular, random seeds and data sampling schemes are not reported from Numerai, such that we cannot replicate the results.

Random seeds are unwanted sources of variability that needs to be controlled [**gundersen2023sources**]. Models trained with all the data will have better performances by design and higher computational costs. Therefore we need to use the same data sampling schemes to fairly train models under the same conditions except that ones we want to change. For GBDT models, model performances also increases with the number of boosting rounds in general, therefore we also need to use an equal amount of rounds to train the models.

Therefore, we create benchmark models using the Ansatz hyperparameters and hyperparameters from Numerai under the **same** random seeds and the **same** data sampling scheme. All the 2132 features are used in training. The target 'target-cyrus-v4-20' are used to train all the models.

We use the data sampling scheme S_2 described in Section 9.2.1 which keep observations with target not equal to the Median value (0.5) in training and trained each model using 25% of data eras regularly sampled. We then obtain 4 benchmark models for each set of hyperparameters (Ansatz and Numerai). The training size of models is fixed to 600, with the training data starting at Era 201.

The Ansatz hyperparameters are found by grid search on different XGBoost models hyperparameters using a subset of features described in Section 9.2.3. The key hyperparameters optimised are: Max Depth: 4, Data Sampling per tree: 0.75, Feature Sampling per tree: 0.75. The learning rate L is given by formula $L = \frac{50}{B}$.

Numerai provided the following hyperparameters [69] for their example model based on LightGBM [70]. The key hyperparameters are: Max Depth: 6, Data Sampling per tree: 1.0, Feature Sampling per tree: 0.1. The recommended the number of boosting rounds $B = 30000$ with learning rates $L = 0.001$, which is interpreted as using the formula $L = \frac{30}{B}$.

As Ansatz hyperparameters uses more shallow trees to build trees than Numerai, it has a much lower memory consumption. On average, for a fixed number of boosting rounds B, the memory consumption of models with Ansatz hyperparameters are only around $30\% - 35\%$ of that of models with Numerai hyperparameters. Computational time is a lower as fewer decision rules are learnt in each. On average, for a fixed number of boosting rounds B, the running time of models with Ansatz hyperparameters are only around $70\% - 80\%$ of that of models with Numerai hyperparameters.

In Figures 27, 28, and 29 in SI, learning curves for Mean Corr and Sharpe ratio of the benchmark XGBoost models are shown under different market regimes.

9.2.1 Sampling data within an era

In this section, we study the impact of different data sampling strategies on model performances. There are different benefits in using different data sampling schemes in model training. The first is to increase diversity of models. Applying data sampling **locally** during tree building are shown to improve diversity of trees efficiently. Similarly, applying data sampling **globally** can enforce our assumptions on the data structure to the model training process to force models to be less correlated to each other by design. Another reason is to reduce computational time in model training, which is critical as newer versions of the Numerai datasets has include more features and data eras.

We consider two sampling strategies S_1, S_2 that can be applied to each data era independently. S_1 is the baseline which uses all the data within an era. S_2 is the method we propose which uses only around half of the data in each era.

- S_1: Using all the stocks within an era
- S_2: Using all the stocks with target $y \neq 0.5$, which means select all the stocks that is not equal to the median value of target. On average we obtain around $45\% - 55\%$ of stocks in each era.

The reason to remove data with target values equal to the Median value is that these data provide little information in learning the ranking of stocks near the tails, which has a bigger impact on the trading portfolio. In practise, only stocks at the top and bottom of the rankings are traded due to transaction costs. Another reason to use S_2 is that it can reduce computational time by half, therefore allowing researchers to train more base models within a deep IL model.

To demonstrate whether the new data sampling scheme S_2 can work well for a wide range of parameter settings for GBDT models, a grid search on two key hyperparameters namely feature sampling per tree and the depth of trees is performed for each data sampling scheme.

- Tree depth: 4,6
- Ratio of feature sampling per tree: 0.1,0.25,0.5,0.75,0.9

The Cartesian product over all combinations of the two hyperparameters gives 10 different hyperparameter settings $G_1, \dots G_{10}$.

The grid search procedure is described in algorithm 15.

Algorithm 15: Grid Search on hyperparameter settings for XGBoost models

Input: Number of boosting rounds B, Training size of Layer 1 $X_1 = 585$, Data embargo $b = 15$
Set starting Era $D_1 = 801$
for $1 \leq j \leq 2$ **do**
  Prepare training data from Era $D_1 - X_1 - b$ to $D_1 - b$ using one of the data sampling schemes S_j
  Set Ansatz learning rate $L = \frac{50}{B}$
  for $1 \leq i \leq 10$ **do**
    Train XGBoost model $M_{i,j}$ with learning rates L, hyperparameter setting G_i.
    Obtain Predictions of models from Era 801 to Era 1070.
  end
end

We run the above procedure for different number of boosting rounds B, with $B = 500, 1000, 2500, 5000$. Each hyperparameter setting is repeated over 4 models using 25% of eras in training data regularly sampled. In Figure 22, we show the risk metrics of the two data sampling schemes, averaged over 10 different hyperparameters settings under different market regimes for different Bs. Sampling with all the data S_1 achieves better Mean Corr in the validation but is not significant in the test period. However, in the test period S_2 achieves a better Sharpe and Calmar ratio.

We then compare the two data sampling schemes under market regimes, which demonstrates the improvement from S_2 in the test period can be mostly attributed to improvement during Bear market. Using sampling S_2 will not lead to a significant deterioration in model performances in bull market and offers valuable hedging benefits during bear market.

Repeating the above analysis using the Ansatz model hyperparameters (Tree Depth = 4 and Ratio of feature sampling per tree = 0.75) only, as shown in Figure 23 demonstrated a even smaller performance gap between model performances of S_1 and S_2.

Therefore, we use S_2 to train train the benchmark models and deep IL XGBoost models.

9.2.2 Using the Ansatz formula to calculate learning rates of GBDT models

We used the Ansatz formula $L = \frac{50}{B}$ to derive the learning rate L for a given number of boosting rounds. Here, we will demonstrate this formula is indeed optimal.

Different approaches are used by researchers to select the learning rates and the number of boosting rounds of GBDT and other ML models for tabular datasets. For small tabular datasets, most researchers would use the default values given by the packages without additional tuning. For example, the Gradient Boosting Regression in Scikit-Learn has default values of 100 boosting rounds and 0.1 learning rate. For larger datasets, researchers would perform hyperparameter optimisation to select the optimal learning rate and boosting rounds.

In most benchmark research papers on ML algorithms for tabular datasets [37, 38, 39], a random search or other Bayesian approach is use to optimise all the hyperparameters of the ML models, ignoring the joint interactions between hyperparameters.

In other research papers [71, 72], either the number of boosting rounds and/or the learning rate is fixed and then the other hyperparameter is optimised. Recent research [61] suggests the learning rate should be determined dynamically to adapt to concept drift in data.

Traditional methods of hyper-parameter optimisation based on random or grid searches are problematic as they ignore the key relationship between the two hyperparameters for GBDT models, namely learning rates and the number of boosting rounds. A blind uniform search on the two dimensional hyperparameter space formed by the number of boosting rounds and learning rate is inefficient.

Intuitively, when learning rate is large, we expect the optimal number of boosting rounds to be small. Similarly, when learning rate is small, the optimal number of boosting rounds should be large. This suggests the optimal learning rate can be written in the form $L = \frac{C}{B} + \mathcal{O}(\frac{1}{B})$ where B is the number of boosting rounds, and C is a constant that depends

on the dataset and other hyperparameters of the GBDT model. For simplicity, we will ignore the higher order terms and assume $L = \frac{C}{B}$.

Using above insights we propose two hypothesis about the learning rates of GBDT models:

Hypothesis (Monotonicity of model performances with respect to learning rate)**.** Consider a GBDT model $\mathcal{M}$ with fixed hyper-parameters except the learning rate l and the number of boosting rounds B. Let $\mathcal{L}_l(B)$ be the loss function of the GBDT model, parameterised by the learning rate and the number of boosting rounds.

For any two learning rates $0 < l_1 < l_2$, we define the minimal value of loss function of model trained with learning rate l_1 obtained at boosting round B_{l_1} as $\mathcal{L}^*_{l_1}(B_{l_1})$. Similarly we define the minimal value of loss of model trained with learning rate l_2 as $\mathcal{L}^*_{l_2}(B_{l_2})$. We would then have $\mathcal{L}^*_{l_1}(B_{l_1}) \leq \mathcal{L}^*_{l_2}(B_{l_2})$. In other words, as we decrease the learning rate, the theoretical optimal model would become better.

We note that this hypothesis is not contradictory to results obtained by random/grid hyperparameter searches in different experiments as we are working within a finite computational budget. The theoretical optimal model may not be able to be reached if we set the upper bound of the number of boosting rounds to a small value. In this situation, the local optimal model within the hyperparameter grid would not always be the model with the smallest learning rate.

Hypothesis (Linear bounds on the number of boosting rounds required to achieve better performances)**.** Consider a GBDT model $\mathcal{M}$ with fixed hyper-parameters except the learning rate l and the number of boosting rounds B. Let $\mathcal{L}_l(B)$ be the loss function of the GBDT model, parameterised by the learning rate and the number of boosting rounds. For any given learning rate $l > 0$, number of boosting rounds B and any constant $c > 1$, we have $\mathcal{L}_{\frac{l}{c}}(cB) \leq \mathcal{L}_l(B)$.

This hypothesis provides us a way to extrapolate optimised learning rates for a given number of boosting rounds to others. This is the basis for the Ansatz learning formula, $L = \frac{C}{B}$.

In algorithm 16 we describe how to search over different learning rates for XGBoost models with a given number of boosting rounds$B$. B is set to 1000,2500,5000,50000. Hyperparameters other than the learning rates are the same as the ones used by benchmark **Ansatz** model.

For each B, we create 5 learning rates based on the Ansatz learning rate $L = \frac{50}{B}$ by considering learning rates larger ($2L$,$4L$) and smaller ($\frac{L}{2}$, $\frac{L}{4}$). In total we have 5 different learning rates including the Ansatz. We train 4 models for each learning rate, where each model is trained using 25% of data eras in the training period, regularly sampled with different start era so the whole dataset is covered.

Algorithm 16: XGBoost models over learning rate

Input: Number of boosting rounds B, Training size of Layer 1 $X_1 = 585$, Retrain Frequency $T = 50$, Data embargo $b = 15$

Set starting Era $D_1 = 801$

Prepare training data from Era $D_1 - X_1 - b$ to $D_1 - b$

Set Ansatz learning rate $L = \frac{50}{B}$

for $1 \leq j \leq 5$ **do**

  Train Layer 1 XGBoost models M_j, with learning rates $l_j = \frac{8L}{2^j}$, other hyperparameters are unchanged.

  Obtain 10 model snapshot predictions from model M_j, taken at boosting rounds $\frac{B}{10}, \frac{2B}{10}, \ldots, B$

end

In Figures 24, 25 and 26, the learning curves of XGBoost models with different learning rates are shown for the risk metrics: Mean Corr, Sharpe ratio and Calmar ratio under different regimes. Learning curves show the value of a metric over different stages of the model training process, indicated by the number of boosting rounds.

In the validation period, models of learning rates of $4L$ demonstrated overfitted behaviour. Models of learning rates of $\frac{L}{2}$ and $\frac{L}{4}$ had a lower performance than the models with Ansatz learning rate L in the validation period, suggesting the model is under-fitted. Models with learning rate $2L$ have similar performances with models with learning rate L but with a larger model variance. Therefore, models with learning rate L is indeed optimal in the validation period.

Models with lower learning rates have a more stable training process. In particular, we observe a monotonic increasing trend of learning curves for learning rate $\frac{L}{4}$ within the computational budget B boosting rounds. This property suggests when we are training large models using a very small learning rate, early stopping is not necessary since we will observe only strictly improving model performances in validation period after removing noise effects. However, using a very small learning rate will require a very long training time, which is infeasible in real applications.

In test period, learning curves of rates $\frac{L}{4}$ and $\frac{L}{2}$ performed slightly better, but only significant for small Bs where $B \leq 2500$. When $B \geq 5000$, the Anstaz learning rate L gives the best performances.

Our Ansatz balances both the need of model training efficiency and stability of training process. It gives the upper bound on the optimal learning rate before there is risk of overfitting in the data. Therefore, we cannot further increase learning rate to make model training more efficient without taking additional risks of model overfitting.

The learning curves of other metrics, such as Sharpe and Calmar ratios are noisy and therefore we do not select learning rates based on those.

9.2.3 Selecting ML models for Temporal Tabular Datasets

For different tabular models introduced in section 3.2, hyperparameter optimisation is performed using data before 2018-04-27 (Era 800). The training and validation set is data between 2003-01-03 (Era 1) and 2014-06-26 (Era 600), with the last 25% of data (Era 451 - Era 600) as the validation set. Due to memory constraints, era sub-sampling is applied during model training. 25% of the eras in the training period is used with sampling performed at regular intervals. The performance of the models in the evaluation period, from 2014-07-04 (Era 601) to 2018-04-27 (Era 800) is then used to select hyperparameters for the tabular models. The Mean Corr and Sharpe Ratio of the prediction ranking correlation in the evaluation period is reported. Due to memory issues for training neural network models, a global feature selection process is used to select 50% of the 1586 features from the V4.1 dataset at the start of each model process by random.

Multi-Layer Perceptron Multi-Layer Perceptron (MLP) models without feature projection are trained with different number of encoding and funnel layers using the architecture described in Section 3.2.

- Number of Feature Eng Layers: 0,1,2,3,4
- Number of Funnel Layers: 1,2,3

Other hyperparameters of the neural network models are fixed in the grid search as follows: (Number of epochs: 100, Early Stopping: 10, Learning Rate: 0.001, Dropout: 0.1, Encoding Neuron Scale: 0.8, Funnel Neuron Scale: 0.8, Gradient Clip: 0.5, Loss Function: Pearson Corr)

In Table 3 shows the performances of MLP models with different network architectures over 5 different random seeds.

The architecture with the highest Mean Corr is the model without feature engineering layers and a standard MLP model with 2 linear layers. When the number of funnel layers equals to 1, the MLP model is equivalent to a (regularised) linear model and has the worst performance. Increasing the number of feature engineering layers does not significantly improve Mean Corr. As model complexity increases, model performances are more varied over different random seeds, suggesting the lack of robustness of deep neural network models.

Feature Eng Layers	Funnel Layers	Mean Corr	Sharpe	Calmar
0	1	0.0159 ± 0.0016	0.8042 ± 0.0631	0.0826 ± 0.0087
	2	$\mathbf{0.0235} \pm 0.0001$	$\mathbf{1.1344} \pm 0.0119$	$\mathbf{0.2692} \pm 0.0201$
	3	0.0223 ± 0.0005	1.0478 ± 0.0372	0.2117 ± 0.0072
1	1	0.0222 ± 0.0003	1.0509 ± 0.0112	0.2118 ± 0.0068
	2	0.0216 ± 0.0003	1.0061 ± 0.0377	0.2021 ± 0.0128
	3	0.0224 ± 0.0003	1.0575 ± 0.0121	0.2212 ± 0.0269
2	1	0.0217 ± 0.0004	1.0176 ± 0.0357	0.2104 ± 0.0178
	2	0.0218 ± 0.0009	1.0346 ± 0.0571	0.2005 ± 0.0076
	3	0.0226 ± 0.0006	1.0754 ± 0.0352	0.2348 ± 0.0242
3	1	0.0224 ± 0.0006	1.0467 ± 0.0402	0.2226 ± 0.0281
	2	0.0221 ± 0.0009	1.0564 ± 0.0441	0.2332 ± 0.0291
	3	0.0217 ± 0.0007	1.0245 ± 0.0414	0.2049 ± 0.0156
4	1	0.0215 ± 0.0006	1.0131 ± 0.0192	0.1980 ± 0.0146
	2	0.0219 ± 0.0010	1.0490 ± 0.0673	0.2229 ± 0.0309
	3	0.0218 ± 0.0017	1.0459 ± 0.0880	0.2513 ± 0.0229

Table 3: Neural Network models between 2014-07-04 (Era 601) and 2018-04-27 (Era 800)

XGBoost Root Mean Square Error (RMSE), the standard loss function for regression problems is used to train the XGBoost models. Early-stopping based on Pearson correlation in the validation set is applied to control the model complexity if needed. A grid search is performed to select the data sub-sample and feature sub-sample ratios of the XGBoost models.

- Max Depth: 4,6,8
- Data subsample by tree: 0.25,0.5,0.75
- Feature subsample by tree: 0.25,0.5,0.75
- L1 regularisation: 0, 0.001, 0.01

- L2 regularisation: 0, 0.001, 0.01

Other hyperparameters of the XGBoost models are fixed as follows: (Number of boosting rounds: 5000, Learning rate: 0.01, Grow policy: Depth-wise, Min Samples per node: 10, Feature subsample by level/node: 1)

Table 4 compares performances of XGBoost models by different data subsample ratios, feature subsample ratios and max depth, mean and standard deviation over 45 models of the 9 combinations of L1 and L2 regularisation each with 5 different random seeds are reported.

Calmar ratio is the performance metric with the most variance, suggesting selecting models based on Calmar ratio is not robust. Mean Corr is the least varied metric between random seeds and therefore we use it for hyperparameter selection.

Models with data sub-sampling ratio of 75% performed better than models with data sub-sampling ratio of 50% and 25%, with a lower variance between model performances over different random seeds also. Models with feature sub-sampling ratio of 75% also performed better. XGBoost models with max depth of 4 performed better than models with max depth of 6 and 8 for each fixed data and feature sub-sampling ratios.

Table 5 compares performances of XGBoost models by different L1 and L2 regularisation and max depth with fixed data and feature sub-sampling ratio of 75%. Mean and standard deviation over 5 models with different random seeds are reported. There are no significant difference between model performances over different L1 and L2 regularisation when other model hyperparameters are fixed. Therefore, we set the L1 and L2 regularisation penalty to be zero when training XGBoost models.

We conclude the optimised hyperparameters as follows: (Number of boosting rounds: 5000, Learning rate: 0.01, Max Depth: 4,Data subsample by tree: 0.75, Feature subsample by tree: 0.75, L1 regularisation: 0, L2 regularisation: 0, Grow policy: Depth-wise, Min Samples per node: 10, Feature subsample by level/node: 1)

Conclusion Increasing complexity of MLP models cannot improve model performances. The best performance is achieved by a standard MLP with two layers, which provides the minimal amount of non-linearity required so that the model does not degenerate to a ridge regression model. As suggested in [73], the performance of over-parameterised models are affected by a myriad of factors including model architecture and training process. It cannot be ruled out that there are other model architectures that can make deep learning models performing better than XGBoost. However, as suggested from research on bench-marking of tabular ML models [39], recent deep learning models for tabular data such as TabNet does not always perform better than MLP models. It is unlikely there are advance neural network architectures that are efficient and performed better than MLP.

XGBoost models performed better than MLP models over a wide range of hyperparameters in the evaluation period. The binned nature of features favours the use of decision trees over neural networks, and this view is shared by different reviews on ML algorithms for tabular data [38, 39]. Moreover, MLP models take longer computational time to train and suffers from memory constraints. Therefore, we do not consider the use of MLP in building deep model ensemble for the Numerai dataset. For similar reasons, other advanced neural architectures are not explored here given their high computational resources requirement. These architectures are also known to have a high variation of performances over random seeds [**gundersen2023sources**] and their hyperparameters are difficult to tune.

Data Sample	Feature Sample	Depth	Mean Corr	Sharpe	Calmar
0.25	0.25	4	0.0242 ± 0.0014	1.2126 ± 0.0992	0.3451 ± 0.1237
		6	0.0225 ± 0.0018	1.1502 ± 0.1034	0.3275 ± 0.0727
		8	0.0187 ± 0.0015	1.0045 ± 0.0904	0.2227 ± 0.0858
	0.5	4	0.0236 ± 0.0014	1.1929 ± 0.0706	0.2804 ± 0.0413
		6	0.0222 ± 0.0014	1.1193 ± 0.0825	0.2495 ± 0.075
		8	0.0189 ± 0.0012	0.9999 ± 0.066	0.23 ± 0.0791
	0.75	4	0.0249 ± 0.0016	1.258 ± 0.1066	0.3501 ± 0.1275
		6	0.0228 ± 0.0013	1.1414 ± 0.0666	0.2974 ± 0.0864
		8	0.0188 ± 0.0023	0.9734 ± 0.1425	0.1848 ± 0.075
0.5	0.25	4	0.0259 ± 0.0009	1.2751 ± 0.055	0.3641 ± 0.0809
		6	0.0248 ± 0.0012	1.2453 ± 0.0768	0.3862 ± 0.1135
		8	0.0217 ± 0.0018	1.1244 ± 0.1076	0.3684 ± 0.143
	0.5	4	0.0267 ± 0.001	1.3394 ± 0.0908	0.4423 ± 0.1279
		6	0.0255 ± 0.001	1.2733 ± 0.0603	0.4521 ± 0.1521
		8	0.0224 ± 0.0011	1.1622 ± 0.063	0.4375 ± 0.1299
	0.5	4	0.0268 ± 0.0011	1.3173 ± 0.0842	0.413 ± 0.0998
		6	0.0255 ± 0.0011	1.2716 ± 0.075	0.4429 ± 0.1468
		8	0.0226 ± 0.0014	1.1566 ± 0.1021	0.4315 ± 0.146
0.75	0.25	4	0.0265 ± 0.0009	1.3146 ± 0.0605	0.4388 ± 0.0731
		6	0.0268 ± 0.0009	1.3439 ± 0.0778	0.6006 ± 0.2169
		8	0.0235 ± 0.0005	1.2071 ± 0.048	0.5044 ± 0.1404
	0.5	4	0.0270 ± 0.0007	1.3345 ± 0.0477	0.4345 ± 0.0665
		6	0.0271 ± 0.0007	1.3469 ± 0.0479	**0.6300** ± 0.1526
		8	0.0241 ± 0.0012	1.234 ± 0.0702	0.4843 ± 0.2099
	0.75	4	**0.0273** ± 0.0006	**1.3624** ± 0.0485	0.4885 ± 0.1032
		6	0.0267 ± 0.0009	1.3373 ± 0.0566	0.5509 ± 0.1314
		8	0.0237 ± 0.0005	1.2369 ± 0.065	0.5501 ± 0.1776

Table 4: XGBoost models with different data subsample ratios, feature subsample ratios and max depths between 2014-07-04 (Era 601) and 2018-04-27 (Era 800)

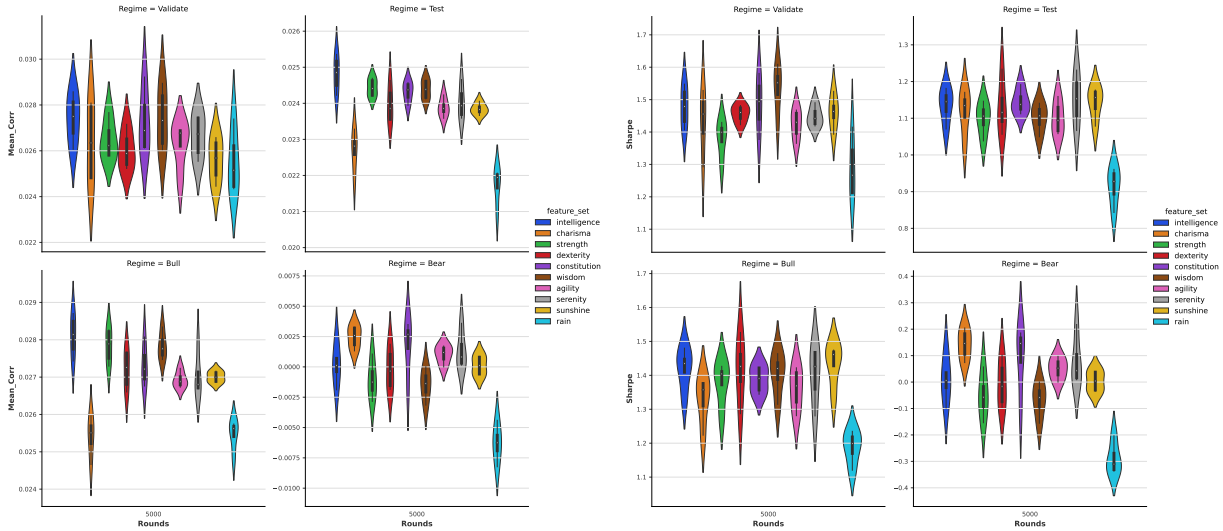
9.3 Additional Results

Max Depth	L2-reg	L1-reg	Mean Corr	Sharpe	Calmar
4	0.0	0.0	**0.0276** $\pm$ 0.0007	**1.3829** $\pm$ 0.0515	0.5305 $\pm$ 0.1077
		0.001	0.0274 $\pm$ 0.0006	1.3764 $\pm$ 0.0485	0.5161 $\pm$ 0.1213
		0.1	0.0268 $\pm$ 0.0008	1.3283 $\pm$ 0.0627	0.4224 $\pm$ 0.0937
	0.001	0.0	**0.0276** $\pm$ 0.0007	**1.3829** $\pm$ 0.0515	0.5305 $\pm$ 0.1077
		0.001	0.0274 $\pm$ 0.0006	1.3764 $\pm$ 0.0485	0.5161 $\pm$ 0.1213
		0.1	0.0268 $\pm$ 0.0008	1.3283 $\pm$ 0.0627	0.4224 $\pm$ 0.0938
	0.1	0.0	0.0271 $\pm$ 0.0002	1.3489 $\pm$ 0.0253	0.4771 $\pm$ 0.0824
		0.001	0.0274 $\pm$ 0.0004	1.368 $\pm$ 0.0358	0.4977 $\pm$ 0.1186
		0.1	0.0274 $\pm$ 0.0004	1.3694 $\pm$ 0.0362	0.4833 $\pm$ 0.0918
6	0.0	0.0	0.0266 $\pm$ 0.0008	1.3354 $\pm$ 0.0453	0.5574 $\pm$ 0.1282
		0.001	0.0267 $\pm$ 0.0011	1.3353 $\pm$ 0.061	0.5422 $\pm$ 0.1697
		0.1	0.0267 $\pm$ 0.0011	1.3201 $\pm$ 0.0638	0.5665 $\pm$ 0.1578
	0.001	0.0	0.0265 $\pm$ 0.0007	1.3347 $\pm$ 0.0441	0.5711 $\pm$ 0.1244
		0.001	0.0268 $\pm$ 0.0011	1.3403 $\pm$ 0.0657	0.5184 $\pm$ 0.1226
		0.1	0.0267 $\pm$ 0.0011	1.3201 $\pm$ 0.0638	0.5665 $\pm$ 0.1578
	0.1	0.0	0.0269 $\pm$ 0.0009	1.3551 $\pm$ 0.0559	**0.6187** $\pm$ 0.1823
		0.001	0.0267 $\pm$ 0.0013	1.3307 $\pm$ 0.0818	0.4871 $\pm$ 0.1164
		0.1	0.0269 $\pm$ 0.0009	1.3638 $\pm$ 0.0563	0.5304 $\pm$ 0.0598
8	0.0	0.0	0.0237 $\pm$ 0.0008	1.2226 $\pm$ 0.0592	0.5127 $\pm$ 0.0875
		0.001	0.0238 $\pm$ 0.0003	1.2445 $\pm$ 0.0623	0.5269 $\pm$ 0.1949
		0.1	0.0237 $\pm$ 0.0004	1.2408 $\pm$ 0.0633	0.5298 $\pm$ 0.0905
	0.001	0.0	0.0238 $\pm$ 0.0007	1.243 $\pm$ 0.0942	0.5657 $\pm$ 0.2615
		0.001	0.0238 $\pm$ 0.0004	1.2383 $\pm$ 0.0254	0.4905 $\pm$ 0.1505
		0.1	0.0238 $\pm$ 0.0003	1.2429 $\pm$ 0.0748	0.6487 $\pm$ 0.1551
	0.1	0.0	0.0235 $\pm$ 0.0006	1.2102 $\pm$ 0.0655	0.6036 $\pm$ 0.2693
		0.001	0.024 $\pm$ 0.0003	1.2461 $\pm$ 0.0552	0.53 $\pm$ 0.1934
		0.1	0.0235 $\pm$ 0.0008	1.2438 $\pm$ 0.1056	0.5426 $\pm$ 0.2102

Table 5: XGBoost models with different L1 and L2 regularisation with fixed data and feature sub-sampling ratios of 75% between 2014-07-04 (Era 601) and 2018-04-27 (Era 800)

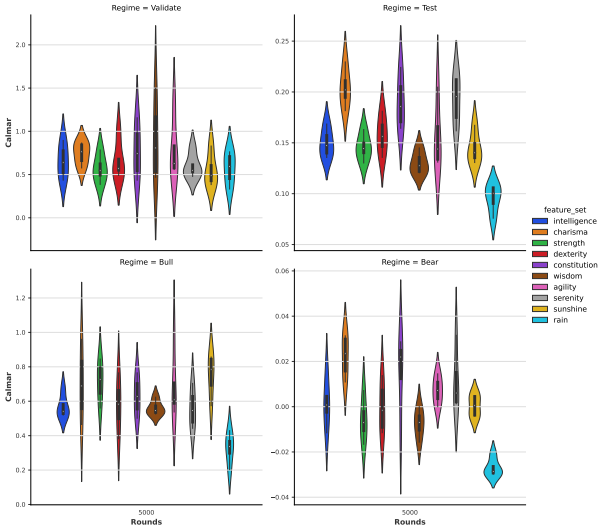
Regime	Strategy	Mean Corr	Sharpe	Max Drawdown
Test	Example Model	0.0264	0.9626	0.2608
	Baseline Model	0.0257	1.0983	0.1839
	Tail Risk Model	0.0022	0.1607	0.2161
	Static Hedged Model	0.0176	1.0507	0.0586
	Dynamic Hedged Model	0.0224	1.2378	0.0415
Bull	Example Model	0.0307	1.2512	0.0693
	Baseline Model	0.0294	1.3962	0.0493
	Tail Risk Model	0.0011	0.0794	0.2161
	Static Hedged Model	0.0195	1.1858	0.0504
	Dynamic Hedged Model	0.0246	1.3731	0.0415
Bear	Example Model	-0.0060	-0.2306	0.2608
	Baseline Model	-0.0018	-0.0831	0.1839
	Tail Risk Model	0.0106	1.0225	0.0027
	Static Hedged Model	0.0033	0.2970	0.0586
	Dynamic Hedged Model	0.0057	0.7733	0.0184

Table 6: Performances of Dynamic Hedged deep IL XGBoost ensemble model based on random feature sampling and V4.2 Example Model from Era 901 to Era 1070 under different market regimes.



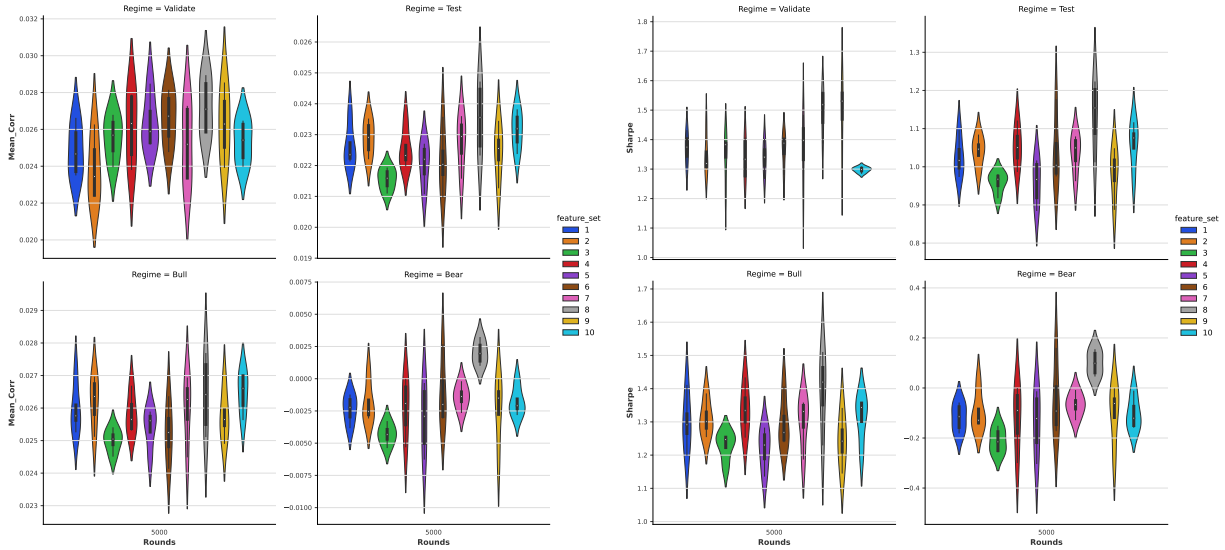
(a) Mean Corr

(b) Sharpe



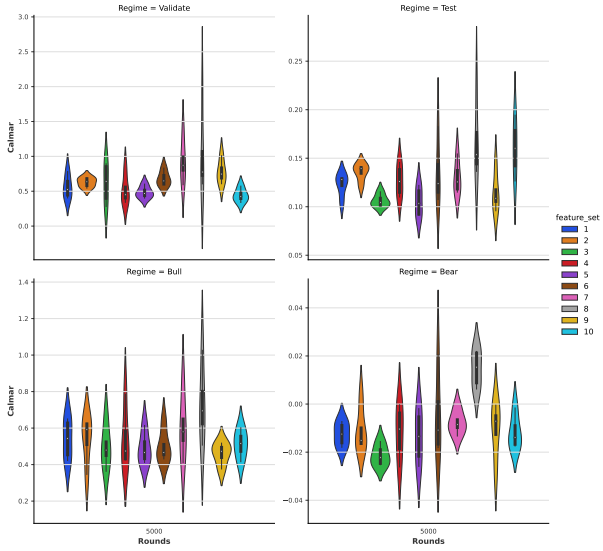
(c) Calmar

Figure 17: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with Jackknife feature sampling under different market regimes.



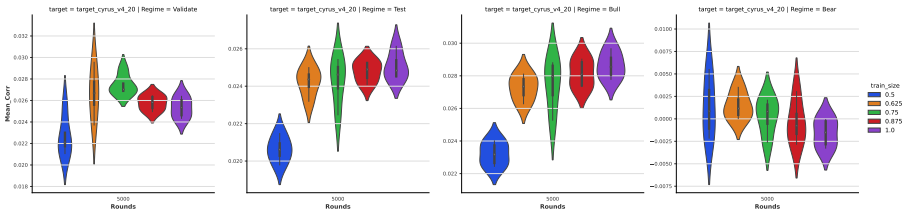
(a) Mean Corr

(b) Sharpe

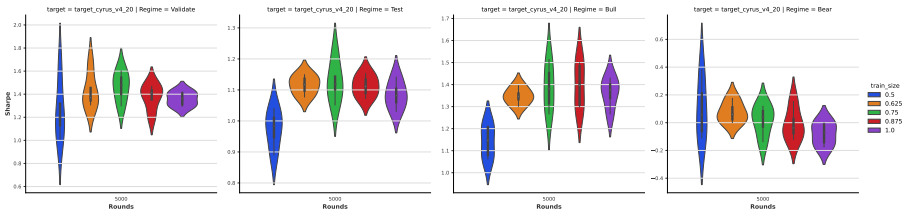


(c) Calmar

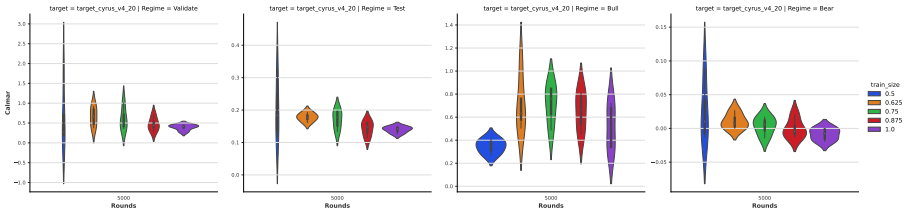
Figure 18: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with random feature sampling under different market regimes.



(a) Mean Corr

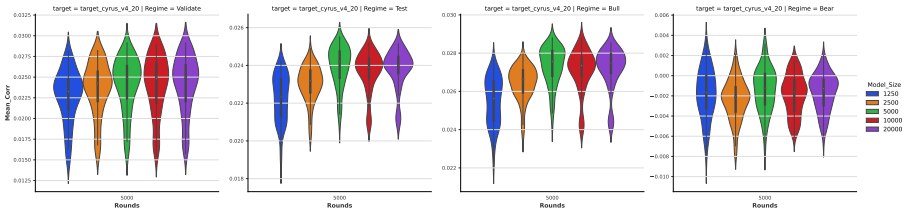


(b) Sharpe

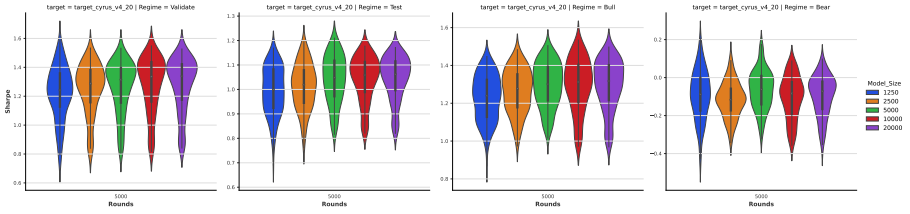


(c) Calmar

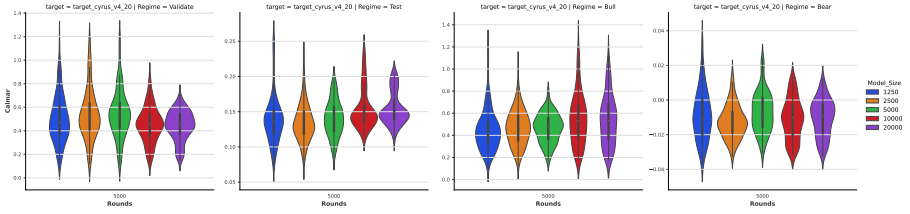
Figure 19: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with different training sizes under different market regimes.



(a) Mean Corr

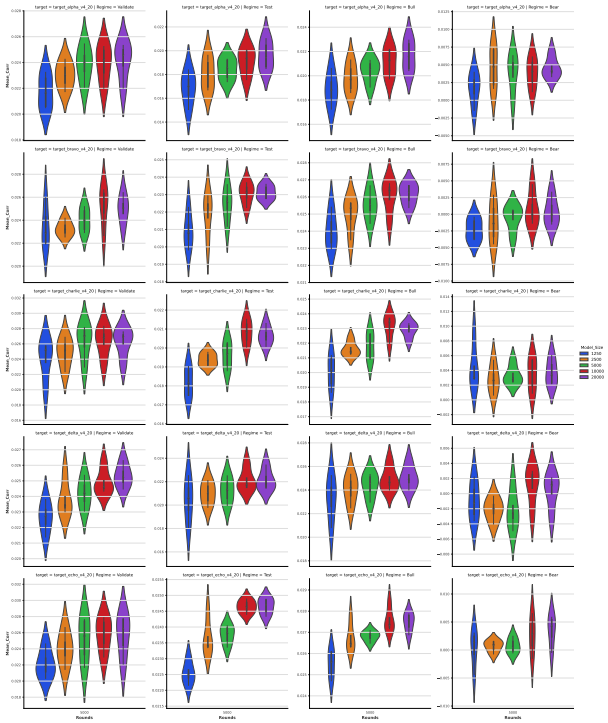


(b) Sharpe

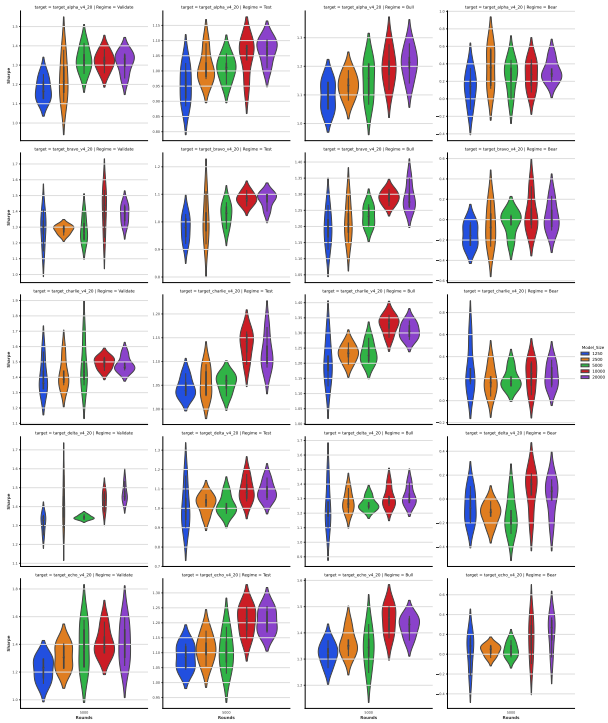


(c) Calmar

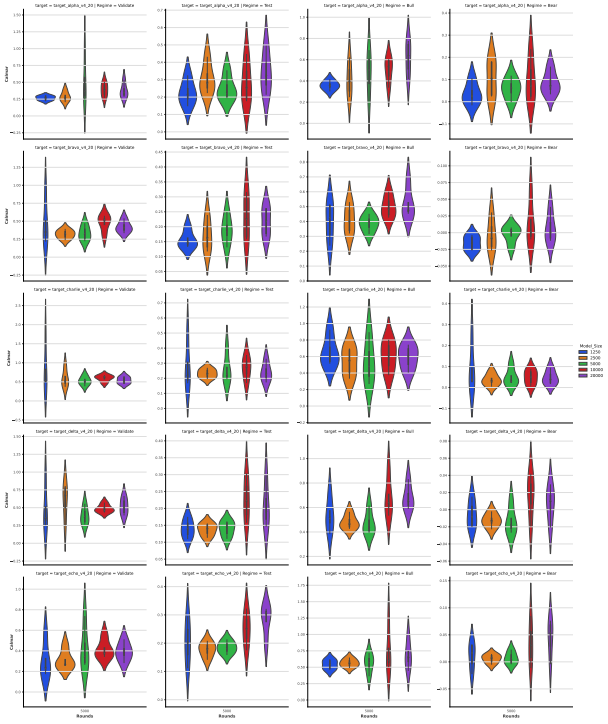
Figure 20: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with different learning rates under different market regimes.



(a) Mean Corr

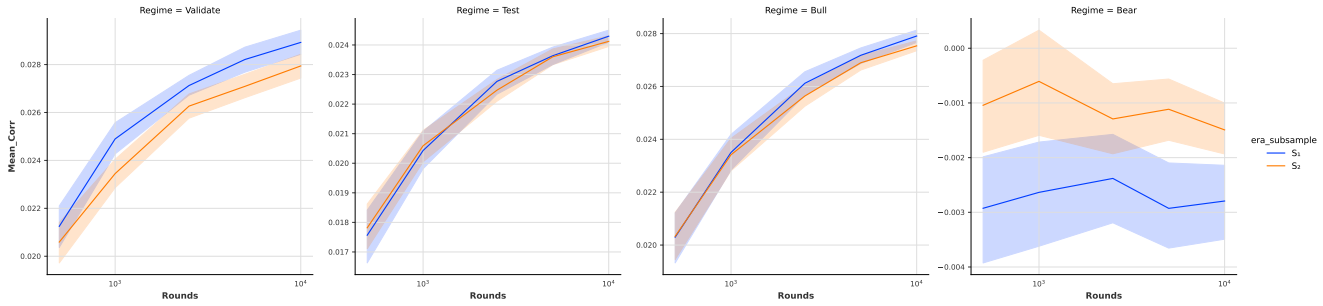


(b) Sharpe

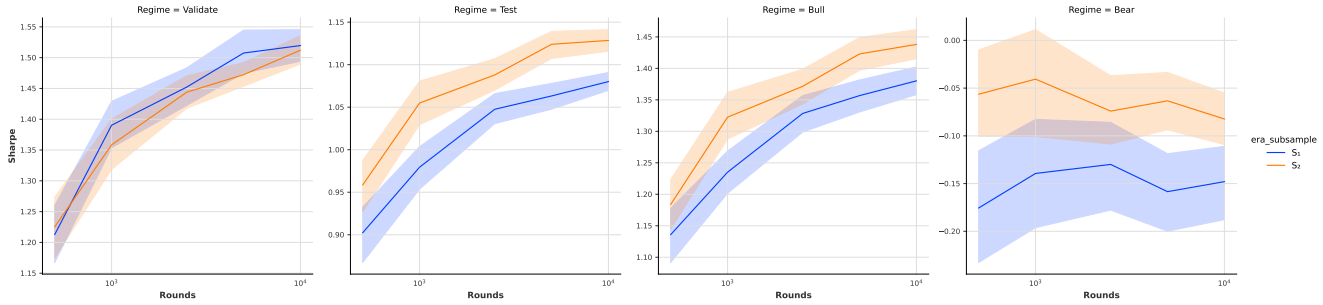


(c) Calmar

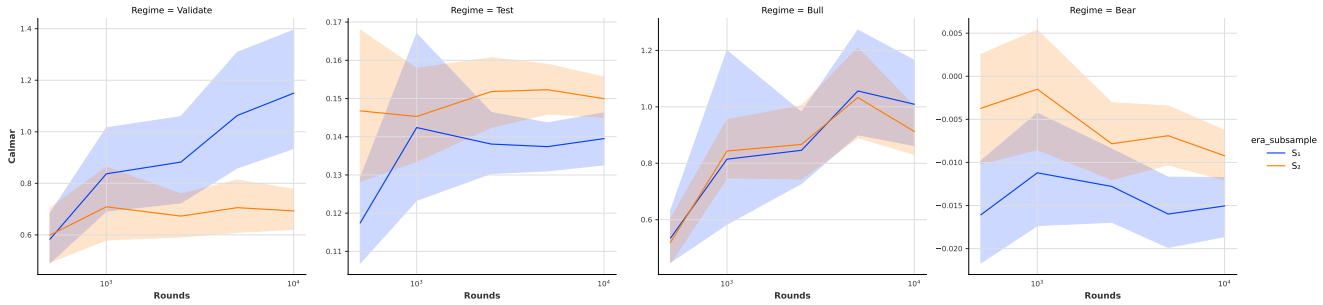
Figure 21: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with different targets and learning rates under different market regimes.



(a) Mean Corr



(b) Sharpe

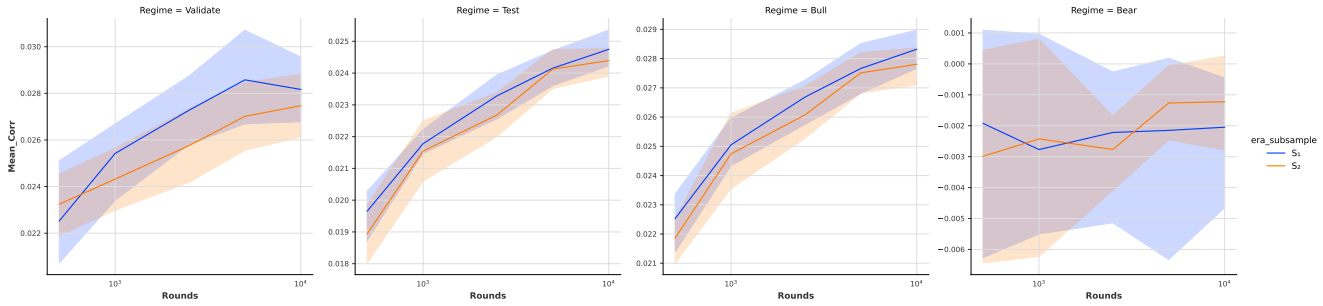


(c) Calmar

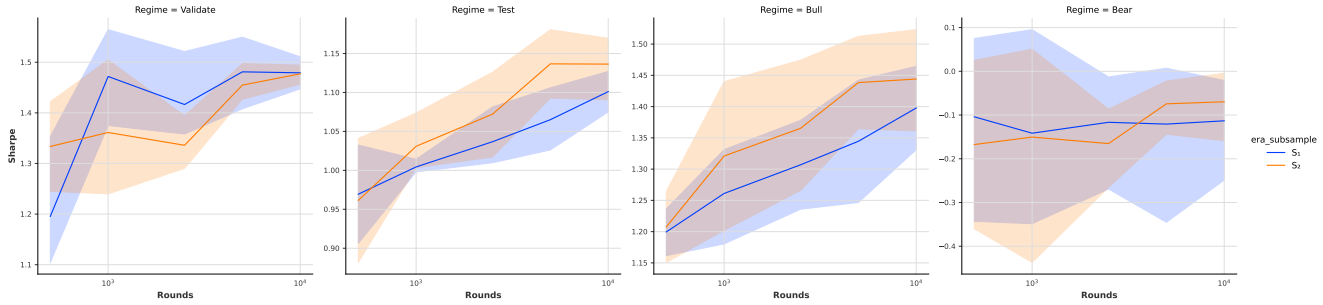
Figure 22: Comparing the performances of two data sampling schemes S_1 and S_2 with different number of boosting rounds $B = 500, 1000, 2500, 5000$ for risk metrics (a) Mean Corr, (b) Sharpe ratio, (c) Calmar ratio, under different market regimes, over 10 different hyperparameter settings

Regime	Strategy	Mean Corr	Sharpe	Max Drawdown
Test	Example Model	0.0264	0.9626	0.2608
	Baseline Model	0.0266	1.1559	0.1646
	Tail Risk Model	0.0016	0.1068	0.3728
	Static Hedged Model	0.0207	1.1337	0.0742
	Dynamic Hedged Model	0.0225	1.2646	0.0330
Bull	Example Model	0.0307	1.2512	0.0693
	Baseline Model	0.0301	1.4305	0.0351
	Tail Risk Model	0.0007	0.0481	0.3728
	Static Hedged Model	0.0227	1.2941	0.0380
	Dynamic Hedged Model	0.0246	1.4305	0.0330
Bear	Example Model	-0.0060	-0.2306	0.2608
	Baseline Model	0.0003	0.0151	0.1646
	Tail Risk Model	0.0081	0.5826	0.0219
	Static Hedged Model	0.0054	0.3409	0.0742
	Dynamic Hedged Model	0.0069	0.4871	0.0312

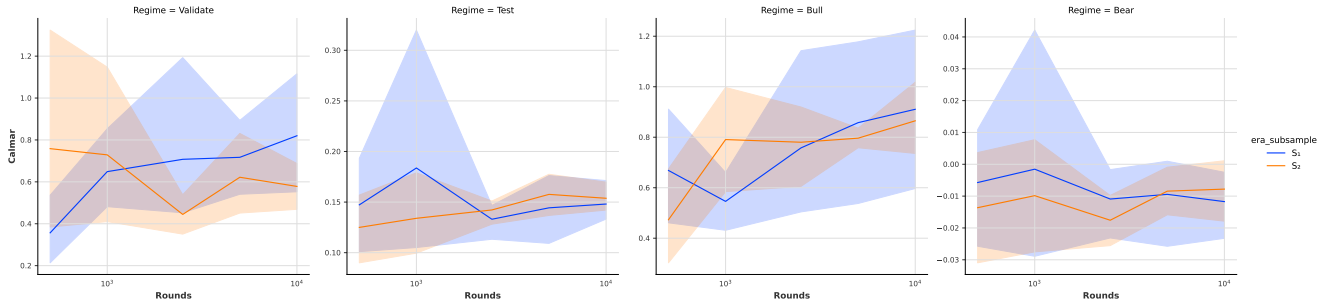
Table 7: Performances of Dynamic Hedged deep IL XGBoost ensemble model based on different training set sizes and V4.2 Example Model from Era 901 to Era 1070 under different market regimes.



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 23: Comparing the performances of two data sampling schemes with different number of boosting rounds $B = 500, 1000, 2500, 5000$ for risk metrics (a) Mean Corr, (b) Sharpe ratio, (c) Calmar ratio, under different market regimes, using the Ansatz hyperparameters Tree Depth = 4 and Ratio of feature sampling per tree = 0.75.

Regime	Strategy	Mean Corr	Sharpe	Max Drawdown
Test	Example Model	0.0264	0.9626	0.2608
	Baseline Model	0.0265	1.1943	0.1562
	Tail Risk Model	0.0015	0.1044	0.1754
	Static Hedged Model	0.0199	0.9978	0.1460
	Dynamic Hedged Model	0.0207	1.0760	0.0871
Bull	Example Model	0.0307	1.2512	0.0693
	Baseline Model	0.0300	1.5073	0.0343
	Tail Risk Model	0.0011	0.0743	0.1754
	Static Hedged Model	0.0227	1.2135	0.0377
	Dynamic Hedged Model	0.0233	1.2755	0.0434
Bear	Example Model	-0.0060	-0.2306	0.2608
	Baseline Model	-0.0001	-0.0053	0.1562
	Tail Risk Model	0.0051	0.2925	0.0743
	Static Hedged Model	-0.0008	-0.0483	0.1460
	Dynamic Hedged Model	0.0015	0.0998	0.0871

Table 8: Performances of Dynamic Hedged deep IL XGBoost ensemble model based on different learning rates and V4.2 Example Model from Era 901 to Era 1070 under different market regimes.

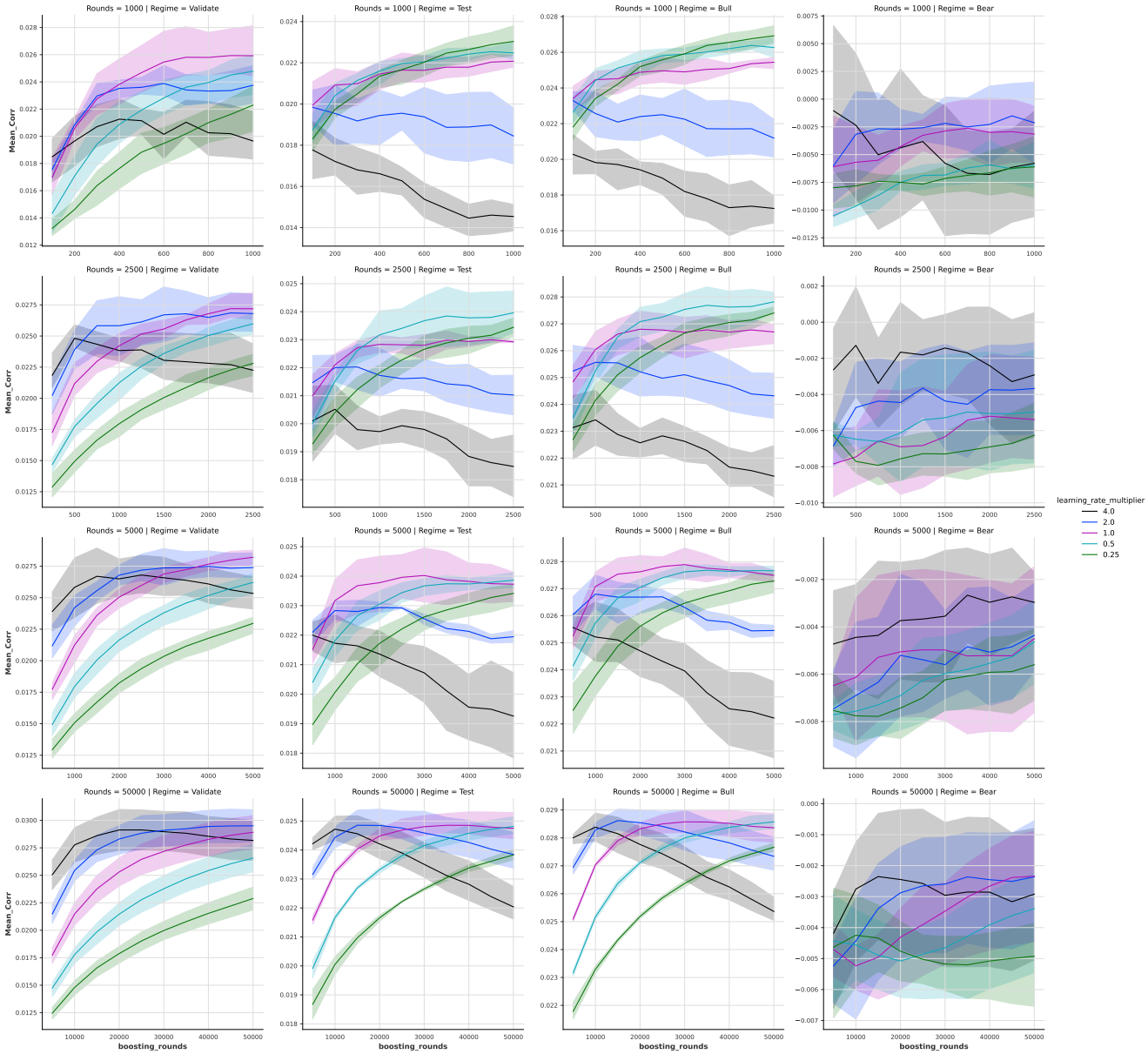


Figure 24: Learning curves of XGBoost models with different learning rates for different number of boosting rounds $B = 1000, 2500, 5000, 50000$ for risk metric Mean Corr under different market regimes

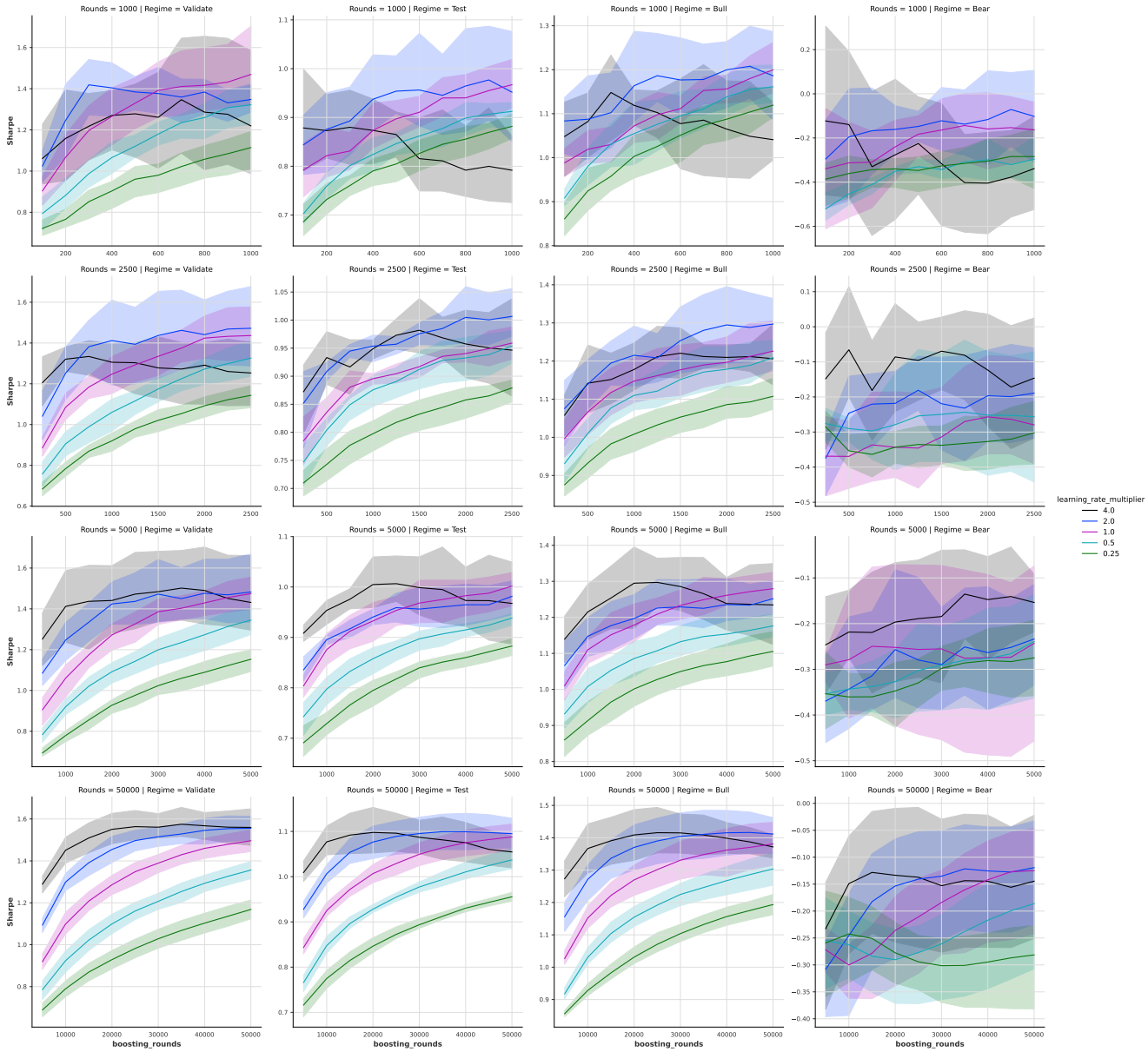


Figure 25: Learning curves of XGBoost models with different learning rates for different number of boosting rounds $B = 1000, 2500, 5000, 50000$ for risk metric Sharpe ratio under different market regimes

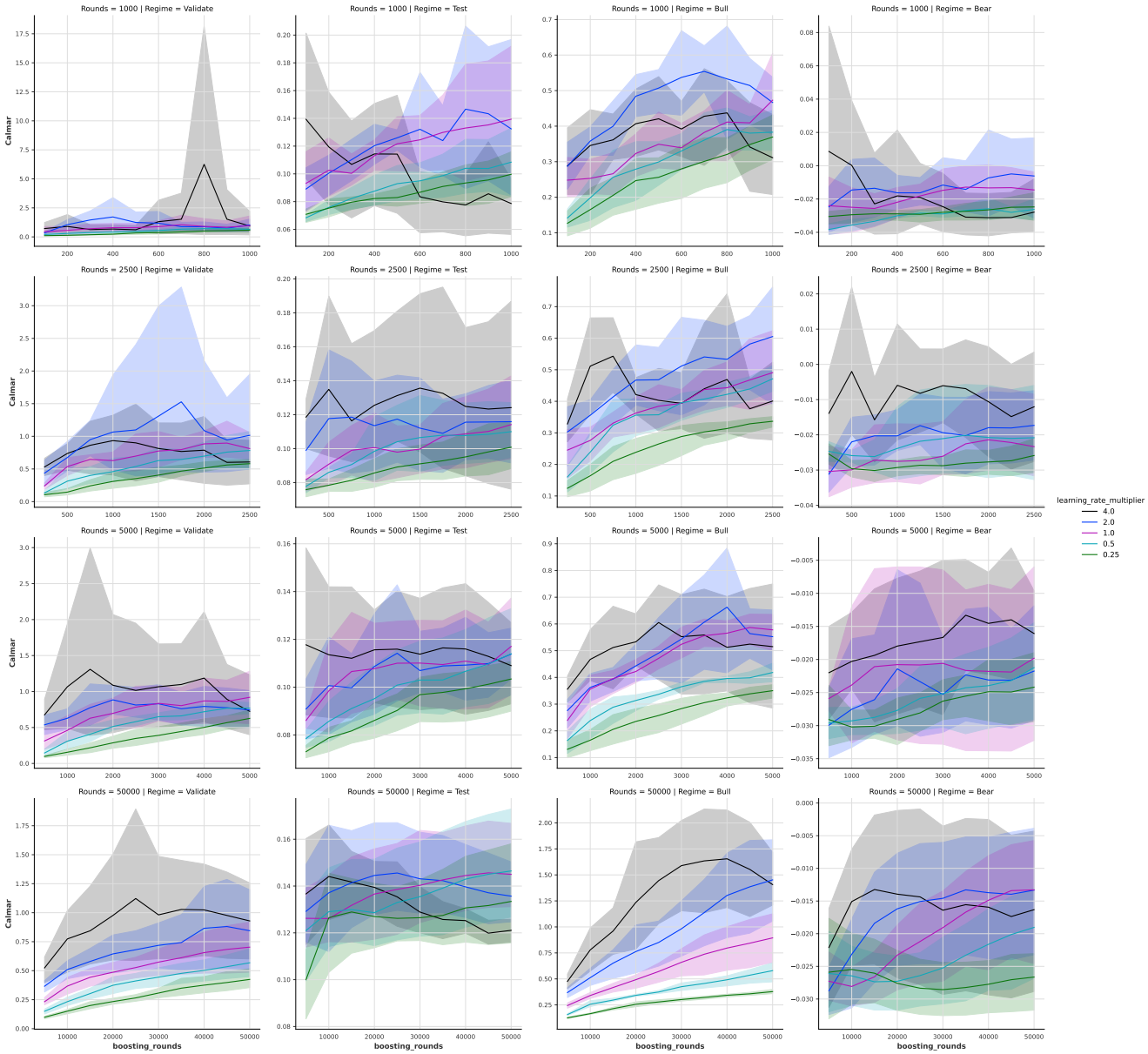


Figure 26: Learning curves of XGBoost models with different learning rates for different number of boosting rounds $B = 1000, 2500, 5000, 50000$ for risk metric Calmar ratio under different market regimes

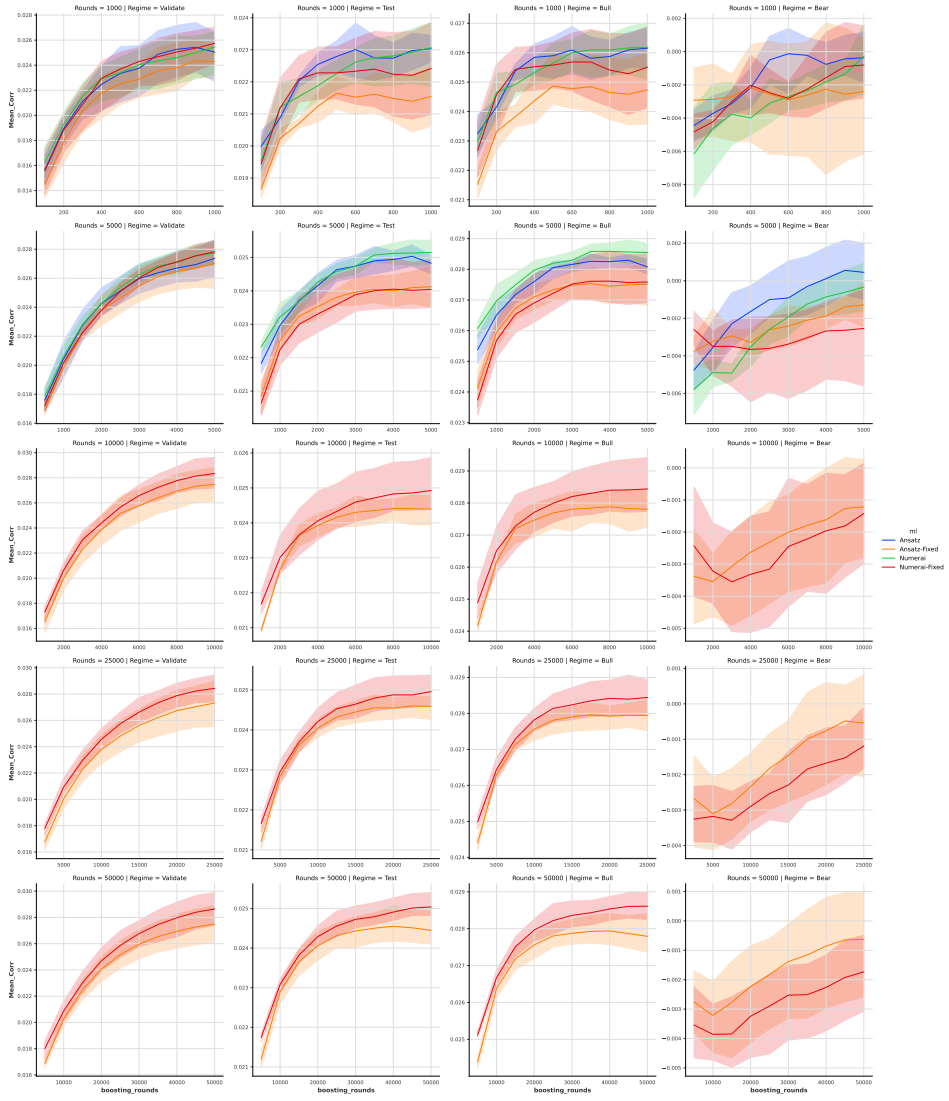


Figure 27: Learning curves of benchmark XGBoost models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$ for risk metric Mean Corr under different market regimes

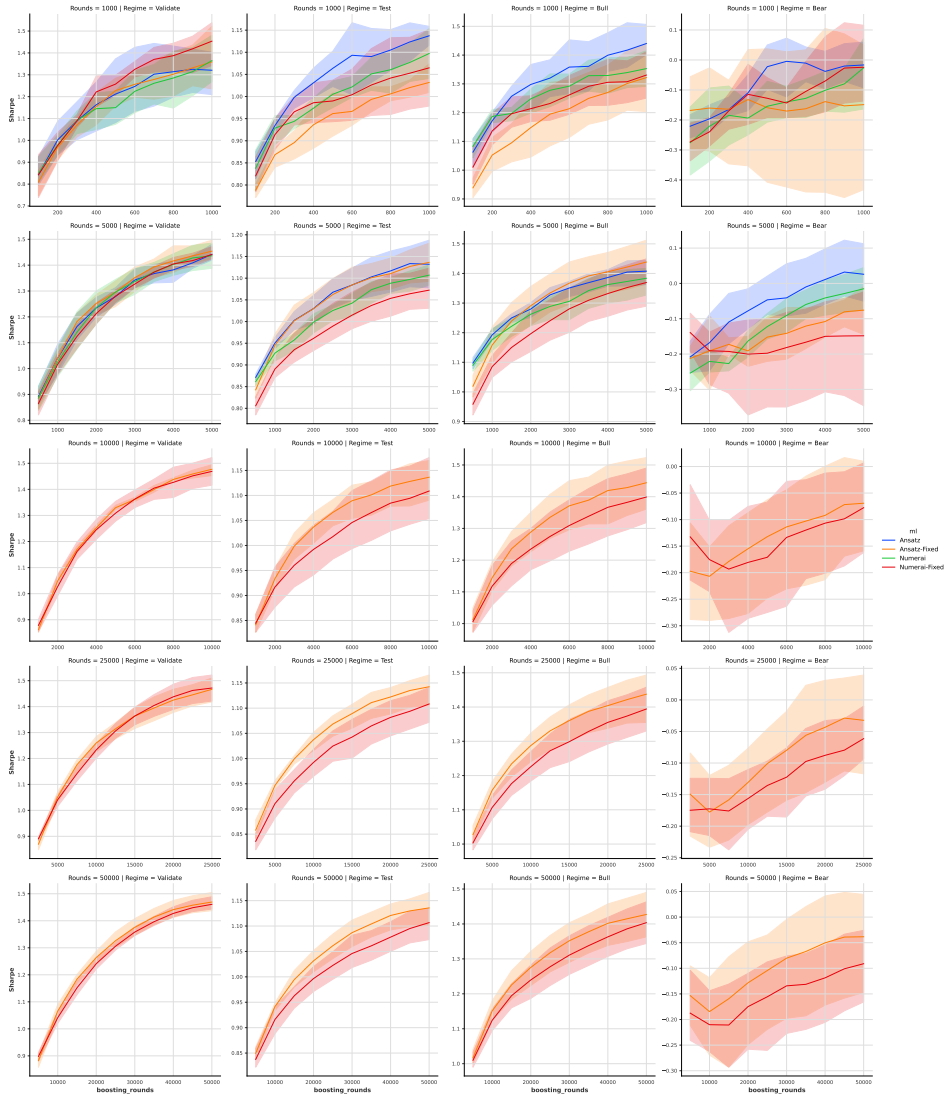


Figure 28: Learning curves of benchmark XGBoost models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$ for risk metric Sharpe under different market regimes

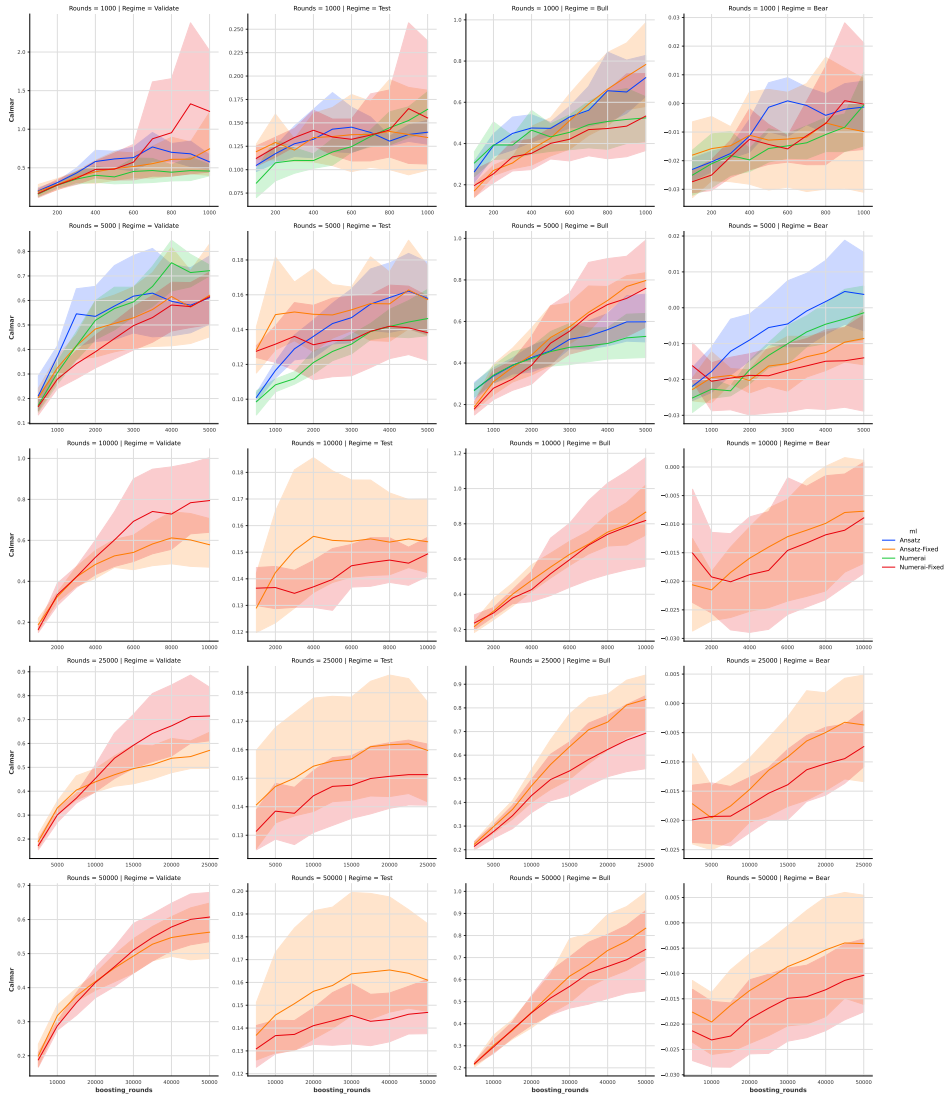
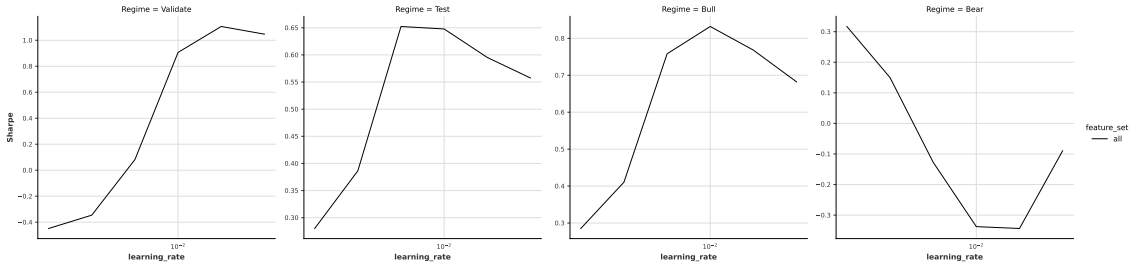
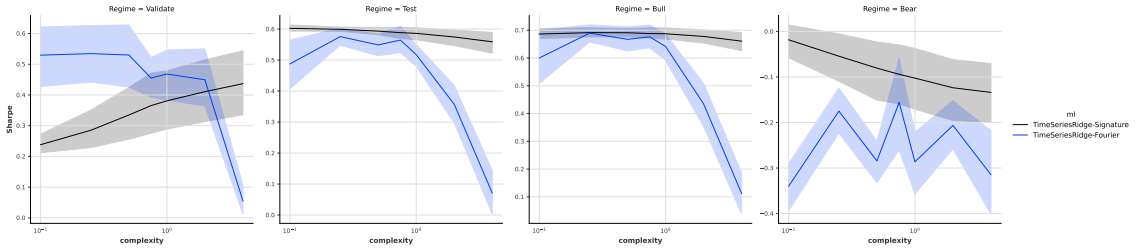


Figure 29: Learning curves of benchmark XGBoost models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$ for risk metric Calmar under different market regimes

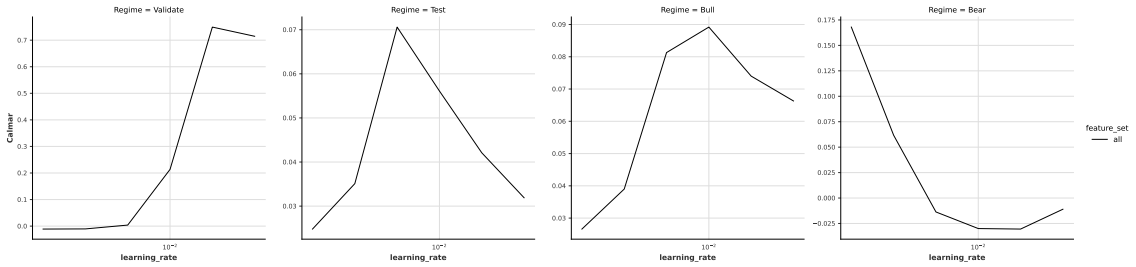


(a) EMA models

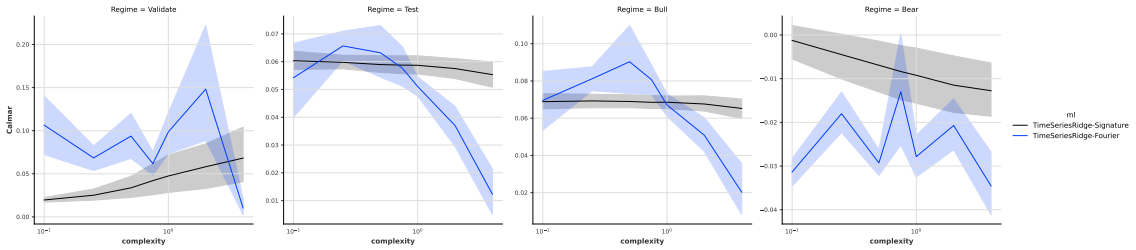


(b) Feature Transform factor-timing models

Figure 30: Sharpe ratio of EMA and Feature Transform factor-timing models under different market regimes



(a) EMA models



(b) Feature Transform factor-timing models

Figure 31: Calmar ratio of EMA and Feature Transform factor-timing models under different market regimes

Regime	Strategy	Mean Corr	Sharpe	Max Drawdown
Test	Example Model	0.0264	0.9626	0.2608
	Baseline Model	0.0247	1.1915	0.1026
	Tail Risk Model	-0.0001	-0.0064	0.5444
	Static Hedged Model	0.0189	0.9937	0.0401
	Dynamic Hedged Model	0.0220	1.2399	0.0457
Bull	Example Model	0.0307	1.2512	0.0693
	Baseline Model	0.0277	1.4344	0.0411
	Tail Risk Model	-0.0017	-0.0990	0.5444
	Static Hedged Model	0.0204	1.0636	0.0401
	Dynamic Hedged Model	0.0234	1.3173	0.0457
Bear	Example Model	-0.0060	-0.2306	0.2608
	Baseline Model	0.0020	0.1219	0.1026
	Tail Risk Model	0.0118	0.8287	0.0148
	Static Hedged Model	0.0078	0.5788	0.0383
	Dynamic Hedged Model	0.0115	0.8475	0.0244

Table 9: Performances of Dynamic Hedged deep IL XGBoost ensemble model based on different targets and V4.2 Example Model from Era 901 to Era 1070 under different market regimes.