Numerai Tournament

Alban Sarfati

MS Financial Engineering, Dauphine 2022
MSc AI, CentraleSupélec 2023
albansarfati@gmail.com

Abstract. In the rapidly evolving financial landscape, the deployment of data-driven approaches to portfolio management has become increasingly critical. Founded in 2015, Numerai is a blockchain-powered, AI-driven hedge fund focused on improving stock market inefficiencies by aggregating thousands of crowdsourced machine learning models to predict the stock market. Leveraging the collective intelligence of data scientists around the world with "obfuscated" financial data, the best submissions are ensembled into a meta model which is used to trade the stock market. The Numerai hedge fund consists of equally weighted long and short portfolios with approximately 500 individual equities in each portfolio, resulting in a long/short portfolio with a beta close to zero. This paper explains the research process that led to participation in the Numerai tournament. Using a cross-sectional strategy, a well-adapted context-aware neural network model is leveraged to generate systematic ranking signals in a sophisticated training configuration. As the field of systematic investments continues to evolve, this research offers a comprehensive blueprint for integrating modern deep learning tools in the investment process to generate alpha.

Keywords: Deep Learning · Encoder · Self-Attention · Forecasting · Cross-Sectional Strategy · Systematic Investment · Stock Market.

1 Introduction

Over the past few decades, the global financial landscape has undergone a paradigm shift. With the explosion of data volumes and the constant evolution of financial markets, traditional portfolio management techniques have been pushed to their limits. In this context of data and decision-making, the merging of technological and informational advantages with financial strategies has emerged as a crucial need.

The stock market, the cornerstone of global finance, is an example of the potential and challenges inherent in managing equities. Representing a vast universe of investment opportunities, the dynamics of this market demand a sophisticated approach. Portfolio managers and investors are faced with the challenge of navigating this universe in order to regularly outperform benchmarks such as the S&P 500. But how can the volume and complexity of available data be effectively exploited to build a solid, high-risk adjusted performance portfolio?

Deep learning, a subset of machine learning, has shown remarkable promise in fields ranging from natural language processing to computer vision. Its ability to discern patterns by finding complex non-linear relationships between inputs and outputs from vast datasets makes it an attractive prospect for finance, particularly in the field of systematic investing. However, applying such learning techniques to finance comes with its unique set of challenges. Financial data often deviates from the IID (independent and identically distributed) assumption, exhibits a low signal-to-noise ratio, and is frequently limited in quantity. Furthermore, financial markets are complex entities in perpetual evolution, a multi-agent system and their available data may sometimes be unrepresentative or even irrelevant.

Numerai ventures into the exciting junction of finance and technology. Their quant fund adapts cutting-edge techniques to the unique challenges posed by managing a universe of stocks. Numerai gives away data so that users all over the world have free hedge-fund quality data to build their machine learning models. The dataset consists of standard financial market data for a universe of roughly 5000 global equities with both traditional and unique features in data that's available over a 10-20 year timeframe. The Numerai data tournament occurs daily; Participants must submit ranked signals for all $\approx$ 5000 global equities in the Numerai data set, forecasting performance over a 20-day period. Only staked submissions will be incorporated into Numerai's Meta Model which ultimately manages the trading for and the portfolios in the Numerai hedge fund. As of July 2023, approximately 5,400 staked models have been submitted for the tournament, with a staked amount totaling $13 million. The Numerai Meta Model is an ensemble of these staked models, the idea that a broad and diverse selection of uncorrelated models will more accurately predict the outcome of a complex financial data science problem. By ranking the performance of all stocks in the universe from best to worst at each given date, the long portfolio would be composed of the best-performing elements of the universe, while the short portfolio would be made up of the worst-performing elements. In this context, regardless of market trends, positive returns are always possible. Numerai has been able to consistently construct a market portfolio where 80%+ of the return is derived from true idiosyncratic (stock selection) risk, rather than the usual factors that drive returns. As of July 1, 2023, the AUM is roughly 320mm.

Momentum strategies are systematic techniques that capitalize on the continued direction of asset returns over time. They take long positions during uptrends and short ones during downtrends. It has been observed that stocks with larger relative returns over the past year tend to have higher average returns over the subsequent year, contradicting the efficient market hypothesis. Such strategies can be categorized into two types: time-series and cross-sectional. Time-series momentum relies on an asset's own historical returns, considering each asset in isolation. It doesn't account for the interactions and predictability between assets. This approach suggests that the signals for individual assets in the portfolio are typically constructed independently of other asset. On the other hand, cross-sectional momentum first quantifies a momentum score for each asset in

the portfolio, then ranks these scores to determine positions. In our context, the time-series strategy is not feasible because the tabular dataset is 'obfuscated', as the stock id is different at each date. However, the specific model can be framed as a cross-sectional strategy by identifying the relative ordering in stock's performance based. The model is optimized according to an approximated Spearman correlation with self-attention based learning which benefits from shared features across assets. The model outputs for each time-step ranking signals by learning rank-order correlation relationships.

2 Related Work

The exploration of data-driven investment strategies has seen significant advancements in recent years, drawing on both traditional financial theories and cutting-edge machine learning methodologies. Machine learning algorithms have been increasingly developed to enhance predictions through data feature extraction and modelling. Yet, deep learning has eliminated the requirement for manual feature engineering, enabling the direct learning of hierarchical feature representations from the data itself. With recent advances in deep learning methods and open source libraries, deep neural networks have been applied to portfolio construction. Yet, the success of a cross-sectional strategy depends critically on accurately ranking assets before portfolio construction. Contemporary techniques perform this ranking step either with simple heuristics or by sorting outputs from standard regression models leading to sub-optimal results as they do not learn the broader relationships across assets. To address this shortcoming, modern cross-sectional strategies have been incorporated sophisticated neural architectures with learning-to-rank losses. In their paper Daniel Poh et al.'s [1] used Multilayer perceptron (MLP) optimized with pair-wise (RankNet) and list-wise (ListNet and ListMLE) loss. The model directly outputs the ranking of individual assets, selection is a thresholding step in which some fraction of assets is retained to form the respective long–short portfolios. However, this basic architecture ignores the differences between the distributions of asset features. To tackle this issue, the authors developed a Fused Encoder Networks [2], based on self-attention mechanism enabling interactions among instruments to be accounted for at the loss level during model training and inference time. Moreover, in [3] they have been using the model developed by Przemysław Pobrotyn et al.'s, in [4]. Przemysław Pobrotyn et al.'s introduced a context-aware neural network model that learns item scores by applying a self-attention mechanism. In this research, they modified the Transformer architecture to work within the ranking framework and obtained a scoring function that, when scoring a single item, takes into account all the other items present in the same list. Daniel Poh et al.'s used this model to encode top/bottom ranked assets, learn the context and exploit this information to re-rank the initial results. These approaches have enhanced the Sharpe ratio and significantly improved various performance metrics sequentially.

3 Problem Definition

In the investment field, one of the most difficult and important tasks in portfolio construction is predicting future performances (i.e. returns). An alternative idea is to predict the relative performance of a group of stocks over time. A portfolio that is long-short would assign positive weights (negative weights) in stocks that are expected to yield the best (worst) returns in the future. For a given long-short portfolio of N_t equities that is re-balanced daily, the realised return of the strategy from day t to $t+1$ is,

$$r_{t+1} = \sum_{i=1}^{N_t} w_t^{(i)} \cdot r_{t+1}^{(i)} \tag{1}$$

where $r_{t+1}^{(i)}$ denotes the daily return for asset i at time $t+1$, $w_t^{(i)}$ denotes the allocation weight for asset i at time t, with $w_t^{(i)} \in [-1, 1]$ and $\sum_i w_t^{(i)} = 0$.

3.1 Learning task

The prediction of ranking signals $X_t^{(i)} \in [0, 1]$ with model f constitutes a learning problem. Given time t, the goal is to construct a learning model f that is able to predict an aggregate signal $\mathbf{x}_t \in [0, 1]^{N_t}$. The spatio input $\mathbf{U}_t \in \mathbb{R}^{N_t \times d}$, where N_t is the number of assets at time t, d is the number of features, and $\mathbf{U}_t(i, k)$ represents the k-th feature of the i-th asset at time t. We want to learn a model f parameterized by θ:

$$\mathbf{x}_t = g(f(\mathbf{U}_t; \theta)) \tag{2}$$

where

$$\mathbf{x}_t = \begin{bmatrix} X_t^{(1)} \\ X_t^{(2)} \\ \vdots \\ X_t^{(N_t)} \end{bmatrix} \tag{3}$$

The ranking signals $\mathbf{x}_t$ are derived using features from the entire stock universe represented as the spatio tensor $\mathbf{U}_t$. g(.) is a transformation function used to improve performance and meet target constraints.

3.2 Signals Construction

The ranking strategy ranks the N_t stocks from best to worst at each given date. The primary scoring function of Numerai measures how well the predicted ranking of the equity matches up with the actual ranking.

1. Signal calculation Presented with an input vector $\mathbf{U}_t$ at t, the strategy's prediction model f computes its corresponding score $\mathbf{y}_t \in \mathbb{R}^{N_t}$ which aims to approximates the future performance for each asset:

$$\mathbf{y}_t = f(\mathbf{U}_t). \tag{4}$$

2. Signal transformation The signals $\mathbf{y}_t$ are scaled to the interval $[0, 1]$ using a min-max scaler, leading to a constant interval across time:

$$\mathbf{y}_t = 2\left(\frac{\mathbf{y}_t - \min(\mathbf{y}_t)}{\max(\mathbf{y}_t) - \min(\mathbf{y}_t)}\right) - 1. \tag{5}$$

3. Security selection Selection is generally a thresholding step in which some fraction of assets is retained to form the respective long-only portfolio. A typical decile-sized long/short portfolio for the strategy (i.e., top/bottom 10%) should be used by Numerai:

$$X_t^{(i)} = \begin{cases} 1 & \text{if } y_t^{(i)} \text{ is in the top decile of } \mathbf{y}_t \\ -1 & \text{if } y_t^{(i)} \text{ is in the worst decile of } \mathbf{y}_t \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

4. Security weighting Each asset i at time t with $X_t^{(i)} = 1$ is assigned a positive weight $w_t^{(i)}$ while those with $X_t^{(i)} = -1$ are assigned a negative weight $w_t^{(i)}$. The weight of each asset is determined based on convex optimization to constraint risk factors such as country, sector and market risk.

4 Model Architecture

Given the complexity of the problem, it is crucial to choose an architecture able to model the relationship between the signals and the features. In their research, Pobrotyn et al.'s [4] proposed a learnable, context-aware, self-attention based scoring function, which allows for modelling inter-item dependencies not only at the loss level but also in the computation of items' scores. They adapt the Transformer architecture to the ranking task, with the encoder-part only, connecting its output directly to the final layer as exhibited in Figure 1. Since the self-attention operation is permutation-equivariant (scores items the same way irrespective of their input order), we obtain a permutation-equivariant scoring function suitable for ranking. Such attention-based encoder offers advantages like enhanced parallel processing, contextual understanding, long-range dependencies, and reduced issues of vanishing or exploding gradients. In our financial context, stock's target for each date are treated as tokens and stock's features as input token embeddings. We denote the length of an input list as N_t, where N_t is the number of assets at time t and the number of features as d. Each item is first passed through a shared fully connected layer of size d_h. Next, hidden representations are passed through an encoder part of Transformer architecture with

N encoder blocks, h heads and hidden dimension d_h. An encoder block in the Transformer consists of a multi-head attention layer with a skip-connection to the input, followed by layer normalisation, time-distributed feed-forward layer, and another skip connection followed by layer normalisation. Dropout is applied before performing summation in residual blocks. Finally, after N encoder blocks, a fully-connected layer shared across all items in the list is used to compute a score for each item. Since the inputs do not possess any inherent sequence, no positional encoding was incorporated into the design.

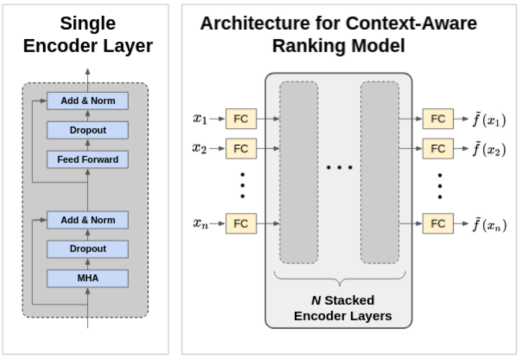


Fig. 1. Model Architecture

4.1 Attention

The model employs a self-attention mechanism to capture patterns and learn long-range dependencies across the list, computing a new, higher-level representation for each item in a list. In general, attention mechanism receives three inputs: queries $\mathbf{Q} \in \mathbb{R}^{M \times d_{\text{attn}}}$, keys $\mathbf{K} \in \mathbb{R}^{M \times d_{\text{attn}}}$ and values $\mathbf{V} \in \mathbb{R}^{M \times d_{\text{V}}}$, these matrices result from the embeddings of the input data. It scale $\mathbf{V}$ based on relationships between $\mathbf{K}$ and $\mathbf{Q}$:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = A(\mathbf{Q}, \mathbf{K})\mathbf{V}, \tag{7}$$

where $A()$ is a normalization function, and M is the number of time steps. A common choice is scaled dot-product attention:

$$A(\mathbf{Q}, \mathbf{K}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_{\text{model}}}}\right). \tag{8}$$

Given input $\mathbf{X} \in \mathbb{R}^{M \times d_{\text{model}}}$, we have:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$$
$$\mathbf{K} = \mathbf{X}\mathbf{W}^K$$
$$\mathbf{V} = \mathbf{X}\mathbf{W}^V$$

$$\text{Attention}(\mathbf{X}) = \text{Softmax}\left(\frac{\mathbf{X}\mathbf{W}^Q(\mathbf{X}\mathbf{W}^K)^T}{\sqrt{d_{\text{model}}}}\right)\mathbf{X}\mathbf{W}^V \tag{9}$$

where $\mathbf{W}^Q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$, $\mathbf{W}^K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$ and $\mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{V}}}$ are parameter matrices to learn. The model learns these parameters that will transform the input $\mathbf{X}$ to queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$. The output of the attention mechanism is a weighted sum of $\mathbf{V}$, where the weights are determined by the interaction between $\mathbf{Q}$ and $\mathbf{K}$. The weights can be interpreted as the relevance between the query and each key. With the help of neural networks, the model will learn the attention distribution of each element relative to the other elements in the same sequence.

Multi-head self-attention achieves greater learning capacity by maintaining parallel and separate attention mechanisms that produce different representations. This increase the flexibility of the model and avoid overfitting as it can be used as ensembles. To address this, the scaled dot-product attention process is performed several times independently to obtain different attention score matrices, which are then concatenated together:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\mathbf{H}_1, \ldots, \mathbf{H}_h]\mathbf{W}^O, \tag{10}$$

$$\mathbf{H}_i = \text{Attention}(\mathbf{X}\mathbf{W}_i^Q, \mathbf{X}\mathbf{W}_i^K, \mathbf{X}\mathbf{W}_i^V), \tag{11}$$

where h denotes the number of head, $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$ and $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{V}}}$ are head-specific weights for keys, queries, and values, and $\mathbf{W}_O \in \mathbb{R}^{(h \times d_V) \times d_{\text{model}}}$ is also a learnable parameter for linear transformation, where typically $d_{\text{v}} = d_{\text{attn}} = d_{\text{model}}/h$. The model can use different ways of learning the dependencies in the list. Based on this design, more complex functions than a simple weighted average can be expressed by a multi-head attention mechanism.

4.2 Layer normalization

Layer normalization is a technique used in the training of deep neural networks to stabilize the learning process. It works by normalizing the inputs across the features for each data sample individually. This characteristic makes layer normalization particularly useful for training models on smaller batch sizes. Layer normalization statistics over all the hidden units in the same layer is computed

as follows:

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$$
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2$$
$$\hat{x}_{ij} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

where m denotes the number of hidden units in a layer and ϵ is a small value.

4.3 Residual Network

The key innovation in ResNet is the introduction of "residual blocks" with skip connections that allow the input of one layer to "skip" one or more layers and be added to the output of a deeper layer. The skip connections mitigate the problem that increasing the number of layers in a neural network will sometimes lead to higher training error.

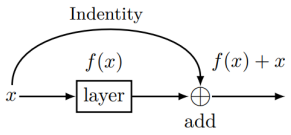


Fig. 2. ResNet

4.4 Position-wise Feed-Forward

Position-Wise Feed-Forward Network (FFN) consists of two fully connected dense layers where the weights are time-distributed. This means each sequence is multiplied by the same weights which is crucial to preserve the temporal dynamics of the input. This block enables non-linear processing and helps prevent vanishing gradients.

$$FFN(x) = FC(Dropout(ReLU(FC(x)))) \tag{12}$$

and FC, Dropout, ReLU are the fully-connected layer, the dropout layer and the ReLU activiation function respectively.

4.5 Output Layer

Thus, the model can be expressed as

$$f(x) = FC(\underbrace{Encoder(Encoder(\dots Encoder}_{N \text{ times}}(FC(x)))))) \tag{13}$$

where

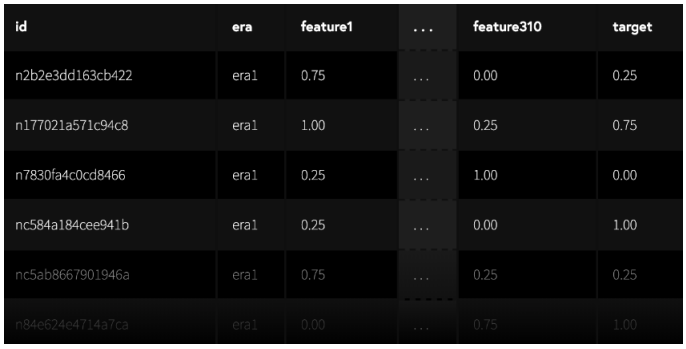
$$Encoder(x) = LayerNorm(z + Dropout(FFN(z))) \tag{14}$$

$$z = LayerNorm(x + Dropout(MultiHead(x))) \tag{15}$$

We thus obtain a general, context-aware permutation-equivariant model for scoring items on a list that can readily be used with any differentiable ranking loss.

5 Dataset Overview

The Numerai dataset is a clean tabular dataset that describes the global stock market over time. At a high level, each row represents a stock at a specific point in time, where id is the stock id and the era is the date. Eras represents different points in time, where feature values are as-of that point in time, and target values as forward looking relative to the point in time. Eras are 1 week apart, with over 1000 historical eras, there are 20 years of historical data. The Numerai dataset is a living dataset, constantly improving through the release of new versions. Released in January 2024, the latest dataset is named Midnight, and is the one used in this research.



id	era	feature1	...	feature310	target
n2b2e3dd163cb422	era1	0.75	...	0.00	0.25
n177021a571c94c8	era1	1.00	...	0.25	0.75
n7830fa4c0cd8466	era1	0.25	...	1.00	0.00
nc584a184cee941b	era1	0.25	...	0.00	1.00
nc5ab8667901946a	era1	0.75	...	0.25	0.25
n84e624e4714a7ca	era1	0.00	...	0.75	1.00

Fig. 3. Dataset overview

5.1 Target

The target of the dataset is specifically engineered to match the strategy of the Numerai hedge fund, it is a measure of future returns (eg. after 20 days) relative to the date. Given their hedge fund is market/country/sector and factor neutral, the target is interpreted as stock-specific returns that are not explained by broader trends in the market/country/sector or well known factors. Apart from the main target, 40 auxiliary targets are provided that are different types of stock specific returns. Like the main target, these auxiliary targets are also

based on stock specific returns but are different in what is residualized (eg. market/country vs sector/factor) and time horizon (eg. 20 day vs 60 days). Building an ensemble of models trained on different targets can also help with performance. The vector target $\mathbf{y}$ at date t for the N_t stocks is $\mathbf{y}_t \in [0,1]^{N_t}$, more precisely target values are binned into 5 unequal bins: 0, 0.25, 0.5, 0.75, 1.0, as one can see in Figure 4.

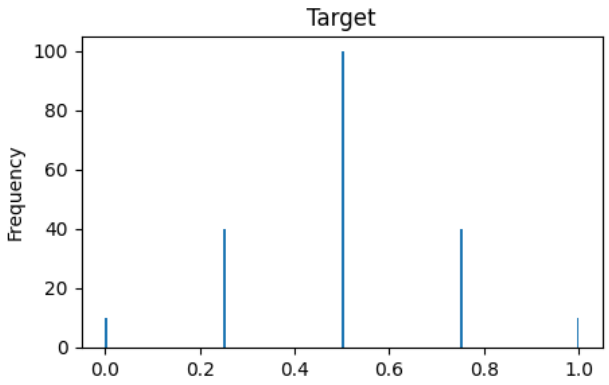


Fig. 4. Distribution of main historical targets

5.2 Features

The spatio tensor $\mathbf{U}_t \in \mathbb{R}^{N_t \times d}$ as in Equation 2, is constructed by mixing both traditional and unique features, resulting in a set of 2376 predictors. These features range from fundamentals like P/E ratio, to technical signals like RSI, to market data like short interest, to secondary data like analyst ratings, and much more. Features can be divided into sets, with the familiar "small", "medium" and "all" sets comprising 42, 705 and 2376 features respectively. There is also a collection of 8 feature groups: Intelligence, Charisma, Strength, Dexterity, etc. Each group contains a different type of feature. For example all technical signals would be in one group, while all analyst predictions and ratings would be in another group. These feature sets are groupings of features which tend to behave similarly. Each feature has been meticulously designed by Numerai to be predictive of the target or additive to other features. They ensured that all features are point-in-time to avoid leakage issues. All of the features have integer values, they are binned into 5 equal bins: 0, 1, 2, 3, 4. This heavy regularization of feature values is to avoid overfitting as the underlying values are extremely noisy. In this research, when it comes to the full feature set, the "medium" and not "all" feature set is used for important reasons such as reducing memory usage and speeding up model training.

5.3 Universe

The universe is evolving over time, hence the number of assets at time t, represented as N_t, can differ from N_{t+1}, resulting in stocks entering or existing the universe. In Figure 5, one can see the growing trend of the universe' size from 2000 stocks (20 years ago) to more than 5000 (now). The pool of investment vehicles is obtained through some rules which are not known.

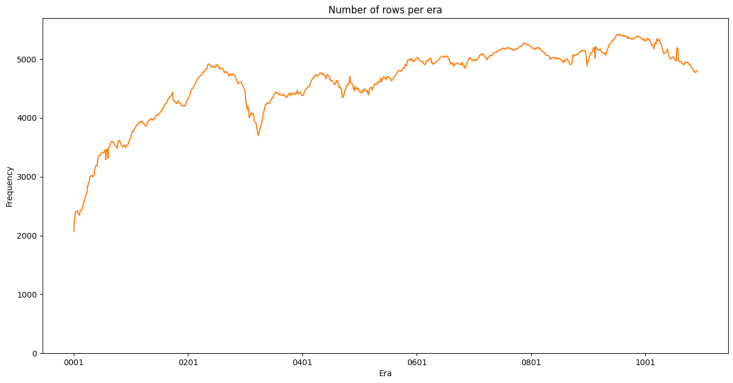


Fig. 5. Historical size of the universe

5.4 Transformation

The features are scaled into $[0, 1]$ by dividing their values by 4. Scaling is a necessary step as it un-biased learning by having inputs and outputs on the same order of magnitude and ensures that each feature contributes equally to the decision-making process. Moreover, neural networks can't process raw integer features directly.

6 Training Setup

6.1 Loss

To optimize a positive relationship between the ranking of model predictions and the ordering of actual future returns, Spearman's correlation has been used to evaluate such a monotonic relationship. A high Spearman correlation indicates that the ranking of predictions aligns well with that of actual future volatility scaled returns. Furthermore, it does not assume a linear relationship between the two variables, which can make it more robust to outliers and non-linearities in

the relationship. The specific Spearman rank correlation being not differentiable, as it operates on ranks rather than the raw data, an approximation has been desgined. Given $\mathbf{y}_{\text{pred}} \in [0,1]^{N_t}$ with $y^{(i)}_{\text{pred}}$ from Equation 5 and the respective targets, $\mathbf{y}_{\text{true}} \in [0,1]^{N_t}$, the approximation is defined as:

1. Computes the means:

$$\boldsymbol{\mu}_{\text{true}} = \frac{1}{N_t} \sum \mathbf{y}_{\text{true}}$$
$$\boldsymbol{\mu}_{\text{pred}} = \frac{1}{N_t} \sum \mathbf{y}_{\text{pred}}$$

2. Centers the values:

$$\mathbf{c}_{\text{true}} = \mathbf{y}_{\text{true}} - \boldsymbol{\mu}_{\text{true}}$$
$$\mathbf{c}_{\text{pred}} = \mathbf{y}_{\text{pred}} - \boldsymbol{\mu}_{\text{pred}}$$

3. Computes the Pearson correlation coefficient:

$$\boldsymbol{r} = \frac{\sum \mathbf{c}_{\text{true}} \times \mathbf{c}_{\text{pred}}}{\sqrt{\sum \mathbf{c}^2_{\text{true}} \times \sum \mathbf{c}^2_{\text{pred}}}}$$

4. Calculates the Squared Difference Penalty:

$$\boldsymbol{sqd} = \frac{\sum (\mathbf{y}_{\text{pred}} - \mathbf{y}_{\text{true}})^2}{N_t \times \sqrt{\sum \mathbf{c}^2_{\text{true}} / N_t}}$$

5. Estimates the Spearman Rank Correlation Loss:

$$\boldsymbol{\rho} = 1 - \boldsymbol{r} + 0.01 \times \boldsymbol{sqd}$$

6. Apply average reduction to the Spearman Rank Correlation Loss:

$$loss = \frac{1}{N_t} \sum \boldsymbol{\rho}$$

This method approximates the Spearman rank correlation, by incorporating both Pearson's correlation and a penalty based on the squared differences between predicted and true values. The use of these two components in the given formula helps the model to simultaneously learn relationships and accuracy. This loss was used with L1 regularization, which is particularly useful for high-dimensional data, as it can encourage weight sparsity, enabling efficient feature selection by reducing the weights of certain features to zero. This is particularly useful in scenarios where a large number of predictors are present, some of which may be correlated. By eliminating these correlated predictors, L1 regularization reduces model complexity while addressing the multicollinearity problem to some extent.

$$loss = loss + \lambda \sum_{j=1}^{p} |w_j| \tag{16}$$

w_j are the weights of the model that the regularization is applied to and λ is the regularization parameter that controls the strength of the L1 penalty. Higher values of λ enforce stronger sparsity in the coefficients.

6.2 Metrics

CORR Numerai uses a special variation of correlation called "Numerai Corr". At a high level, this metric is designed to be a good proxy for actual portfolio returns if the predictions were used in live trading.

```
def numerai_corr(preds, target):
  ranked_preds = (preds.rank(method="average").values - 0.5) /
      preds.count()
  gauss_ranked_preds = stats.norm.ppf(ranked_preds)

  centered_target = target - target.mean()

  preds_p15 = np.sign(gauss_ranked_preds) *
      np.abs(gauss_ranked_preds) ** 1.5
  target_p15 = np.sign(centered_target) *
      np.abs(centered_target) ** 1.5

  return np.corrcoef(preds_p15, target_p15)[0, 1]
```

First the predictions are gauss ranked. This is done to match the live trading process where all model predictions are standardized this way before being ensembled together in the meta model. Then the main target is centered around 0, to match the gauss ranked predictions which are now centered around 0. Finally both the gauss ranked predictions and the centered target are raised to the power of 1.5 before calculating the Pearson correlation. This accentuates the tails as hedge fund tends to only trade the stocks with highest or lowest predicted returns. Consequently, prediction's distribution does not matter and the score depends more on the tails than a typical rank-correlation.

MC Model Contribution (MC) for the Metal Model (MMC) or Benchmark Models (BMC) is the covariance of a model with the main target, after its predictions have been neutralized to the Meta Model or Benchmark Models. This metric tell us how the unique component of a model contributes to the correlation of a specified Model.

```
def contribution(predictions, meta_model, live_targets):
   meta_model, predictions = filter_sort_index(meta_model,
       predictions)
```

```
    live_targets, predictions = filter_sort_index(live_targets,
        predictions)
    live_targets, meta_model = filter_sort_index(live_targets,
        meta_model)

    p = gaussian(tie_kept_rank(predictions))
    m = gaussian(tie_kept_rank(meta_model.to_frame()))

    neutral_preds = orthogonalize(p, m)

    live_targets -= live_targets.mean()

    mmc = (live_targets @ neutral_preds) / len(live_targets)

    return mmc
```

First the predictions are normalized, then the Meta Model (or Benchmark) is normalized, the submissions are neutralized to the Meta Model (or Benchmark) and finally the covariance of the neutral submission with the target is found. By neutralizing the model's predictions by the Meta Model or Benchmark Models, the remaining orthogonal component's covariance with the target is that model's contribution. One key element to understand is that MMC over the validation period may not be truly indicative of out-of-sample performance. Indeed, the Meta Model over the early validation period did not have access to newer data/-targets. This is why it is helpful to measure the contribution of a model to a Numerai's benchmark model. This is an important metric to track because as it tells how additive the model is to Numerai's known models and, by extension, how additive it might be to the Meta Model in the future.

Payout After the 20 days of scoring for each live submission, models with positive scores are rewarded with more NMR (Numerai crypto-currency), while those with negative scores have a portion of their staked NMR burned. The payout is primarily a function of CORR and MMC scores:

$$payout = 0.5 * CORR + 2 * MMC \tag{17}$$

LearnPayout As the MMC over the validation period is misleading, BMC was computed to measure the contribution of the model to an equally-weighted ensemble of Numerai's benchmark models for version 43 (midnight dataset). A variation of the payout metric has been designed, replacing MMC with BMC and giving more weight to CORR, as it is the only metric that is robust over time:

$$lpayout = CORR + 2 * BMC \tag{18}$$

6.3 TS-CV

To ensure that the model generalizes well to unseen data, a Time-Series Cross-Validation (TS-CV) with rolling window has been implemented to respect the inherent temporal order of the data. As the data has a 20-year history, it was necessary to reduce the history to speed up the cross-validation process. Moreover, the Meta Model predictions are only available from era 888, so the MMC metric can only be calculated from era 888, therefore the first validation fold has been chosen to starts at era 888. In this context, to adapt to changes in the universe and market conditions, a rolling window of the most recent data was used. In the first fold, the model is trained on the last 3 years (156 eras) of data before era 888, validated on the next 3 months of data (13 eras) and tested on the following next 3 months of data. The next folds involve on-the-fly learning where the model is updated continuously as new data becomes available. On-the-fly learning allows the model to adapt to new patterns or changes in the data distribution over time, making it particularly useful for stock market applications where updating knowledge in real-time is necessary. Therefore for each new fold, the learned model (from the last fold) is trained on 3 months data, validated on 3 months data and also tested on 3 months data. This constitutes a total of 15 folds, allowing for multiple train-validation-test splits, where the validation set is always ahead of the training split and the testing set is always ahead of the validation split. After the first fold, each training set length remains constant, stabilizing the training-process across folds. To prevent leakage, which takes place when the training (validation) set contains information that also appears in the validation (testing), the training-validation-testing sets are separated by a gap of 4 eras as the labels are derived from 20 days. By fitting and inferring on different time periods, we get a better sense of the model's average performance. In this configuration, the validation and testing sets are constant and do not overlap between the folds, with a total validation period from era 888 to 1082 and a total testing period from era 905 to 1092. As shown in Figure 6, at the first fold, the model is trained from era 728 to era 883, validated on era 888 to era 900 data and tested on era 905 to era 917, for the next iteration, the same model (with learned weights) is trained on data from era 884 to era 896, validated on era 901 to era 913 data, and tested on era 918 to era 930 and so on to the end of the historical data (15 folds).

6.4 Optimization

The Context-Aware Learning to Rank model is trained with the Adam optimizer, leading to the minimization of the approximate Spearman loss function from 6.1. Backpropagation is performed for a maximum of 20 epochs, during which, for a given training set, there is a train-validation-test split. Early stopping is used to prevent model overfitting, this is triggered when the payout metric from 6.2 on the validation set did not improve for 5 consecutive epochs. Using TS-CV, during the first fold the model (with random weights) is trained on data from era era 728 to era 883, if at some point the payout metric on the validation data (era

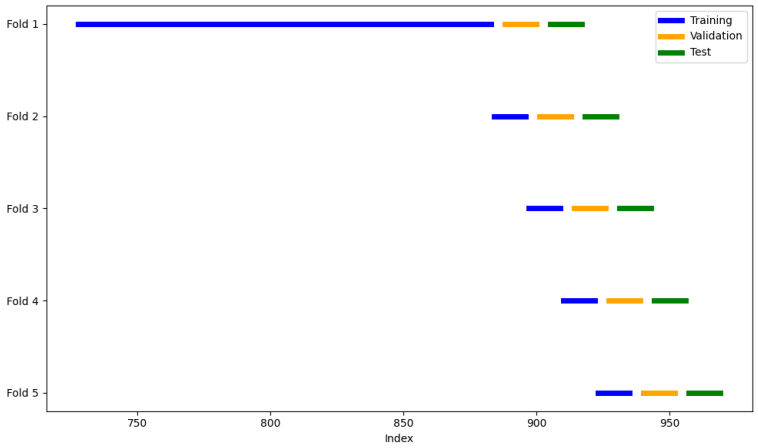


Fig. 6. Time-Series Cross-Validation sample

888 to era 900) did not improve for 5 consecutive epochs, then the model weights at epoch x are saved (epoch x is the epoch for which the validation payout is the biggest), we quit the training process and enter the testing process, where the model is loaded with the weights of epoch x to do the inference on the testing data from era 905 to era 917 leading to the computation of test scores/metrics. For the second fold, the model is loaded with the weights of epoch x of fold 1 (no more random weights) and trained on data from era 888 to era 900, validating on data from era 901 to era 913 the model with the best weights is saved (always according to validation payout metric), while fold 1 saving is deleted, then the latter infers the testing data from era 918 to era 930 and it goes on.

Dynamic padding A custom collate function has been designed to facilitates dynamic padding, ensuring that all sequences in a batch have the same length. Dynamic padding refers to the process of padding lists (with 0) within each batch to the length of the longest list in that batch, as opposed to static padding where all lists in the dataset are padded to a fixed length predefined before training. This approach is more efficient because it minimizes the amount of unnecessary padding, which can save memory and reduce computational overhead.

Masking As the size of the universe evolves over time, masking has been used in the loss, by setting targets and outputs coming from padded list to 0 during loss computation. This approach ensures that only meaningful data contributes to the loss, which is essential for training the model effectively.

Shuffling Stock market data is inherently sequential and time-dependent. The order of the data points represents the chronological order of market events,

which is crucial for predicting future stock movements. Shuffling the data would disrupt this order, making it impossible to capture the time dependencies and patterns essential to accurate forecasting. In order to preserve the causal relationship between past and future data points, shuffling was deactivated during model training.

Batch-size To learn cross-sectional representation the learning batch-size was set at 2. A small batch-size results in more frequent gradient updates, which can help escape local minima in the loss landscape and potentially leading to better generalization. Moreover, it doesn't require as much memory as large batch-size, which would have been a problem with Numerai's very large dataset. For validation and prediction, the batch size was set at 1 to ensure accuracy and simplicity in the model evaluation and prediction process. This approach allows each data instance to be processed and evaluated independently, which is crucial for maintaining the integrity of temporal relationships and ensuring that predictions are made accurately on an individual basis.

Hidden dimension Simpler models are designed by using just 16 neurons (for the full feature set) or 8 neurons (for a single feature group) per fully-connected layer, to improve calibration and prediction accuracy, resulting in easier alignment with financial data and reduced errors when applied beyond the sample.

Attention heads The use of multi-head attention (h) was set to 4 to take advantage of the benefits of 4 heads, as discussed in section 4.7, such as capturing information in the sequence under different aspects.

Seed Due to the stochastic nature of neural networks, the seed has been set to the default value 42 for reproducibility and consistent results.

The number of encoder blocks is treated as a tunable hyper-parameter. A dropout regularization is used as an additional safeguard against overfitting, the dropout rate is similarly treated as a hyperparameter. The coefficient of the Lasso penalty (l1-regularization) is likewise considered a hyperparameter to find the ideal extent of shrinkage applied to the coefficients of the model. Finally, the learning rate which is the most important hyperparameter, as it determines the step size at each iteration while moving towards a minimum of the loss function, is also calibrated over model learning.

6.5 Hyper Optimization

A bayesian hyper-optimization was performed, as an outer optimisation loop, to select the best hyperparameters, based on the TS-CV discussed in 6.3.

Parameters By using TS-CV, we get a better idea of how the different hyperparameter values behave over different time periods, enabling us to select values that generalize well. Hyperparameter optimization is performed with 30 iterations of bayesian search, using the search range as shown in Table 1. As there is a notion of time in financial data, and very often the data in the training set and the data in the test set do not follow the same distribution, this leads to huge out-of-sample errors if the model is too complex. To avoid this, the number of encoder blocks and the hidden size of the network has been limited.

Table 1. Hyperparameter Search Range 1

Hyperparameters	
Encoder block(s)	1, 2
Dropout Rate	[0.1, 0.3]
Learning Rate	[0.0001, 0.01]
L1 reguralization	[0.001, 0.01]

The L1 reguralization range is lowered to $[0.0001, 0.001]$ if only one feature group is used to train the model. As a smaller, less complex dataset might inherently require less regularization to prevent overfitting.

K-fold Score To select the best model through the search iteration, a k-fold score, corresponding to the Sharpe ratio of the LearnPayout metric over the test folds, of each candidate model is calculated as follows

$$k_{\text{fold\_score}} = \frac{\frac{1}{15}\sum_{i=1}^{15} \text{lpayout}_{\text{fold}i}}{\sqrt{\frac{1}{15}\sum_{i=1}^{15}(\text{lpayout}_{\text{fold}i} - \bar{\text{lpayout}})^2}} \tag{19}$$

where $\text{payout}_{\text{fold}i}$ is the LearnPayout metric during the test of the candidate model on fold i and $\bar{\text{payout}}$ is the average of the test LearnPayout metric over all folds. The test LearnPayout metric comes from the inference of the model whose weights come from the epoch where it's validation LearnPayout metric was the maximum (as explain in 6.4 Optimization). Consequently, the best model is the one with the highest k-fold score, meaning that this performance is stable across the various time-based validation splits.

7 Out-of-Sample Evaluation

7.1 Back-testing Details

Train-Test Scheme

The models, calibrated by the hyper-optimization step, are configured with the TS-CV as explained in section 6.3 and 6.4 . In this approach, the training

window expands as new data becomes available, enabling the model to continuously learn from the most recent data. Re-training takes place every three months, a frequency that balances the exploitation of new data with computational efficiency. This re-training ensures that the model remains up to date with the latest trends and patterns, maximizing its predictive performance for the following three-month validation and testing period.

Models

Each model resulting from 6.5 Hyper-Optimzation can be trained on the main target (cyrus20) with the full feature set, on an auxiliary target (teager20) also with the full feature set or on the main target with only one group of features. The result is 10 hyper-optimization projects, each comprising 30 trials. For each projects, the top model based on the k-fold score is saved and backtested. In addition, an ensemble model was designed by equally weighting the predictions of the best combination between the 10 models. Looking at the network architectures resulting from the hyper-optimization, it's clear that the models balance between low complexity and prediction quality, with a maximum of 15k parameters. This compact design not only speeds up computation, but also reduces the probability of unstable-training and over-fitting. Fewer parameters in this Context-Aware Learning to Rank model lead to better generalization, especially considering the inherent complexities and challenges associated with financial data.

7.2 Results and Discussion

	CORR mean	MMC mean	BMC mean	lpayout mean	lpayout sharpe
Cyrus20FullFeatures	0.009	0.007	0.006	0.022	1.33
Teager20FullFeatures	0.005	0.005	0.005	0.015	1.17
Cyrus20Intelligence	0.004	0.002	0.002	0.007	0.49
Cyrus20Wisdom	0.009	0.008	0.008	0.025	1.81
Cyrus20Charisma	0.006	0.003	0.003	0.011	0.63
Cyrus20Dexterity	0.004	0.003	0.002	0.008	0.55
Cyrus20Strength	0.007	0.006	0.006	0.018	1.10
Cyrus20Constitution	0.005	0.003	0.004	0.011	0.80
Cyrus20Agility	0.005	0.003	0.003	0.010	0.75
Cyrus20Serenity	0.003	0.004	0.003	0.010	1.25
Ensemble	0.014	0.010	0.011	0.035	2.37

Table 2. Backtest statistics

Looking at the table showing backtest statistics over the test period (era 905 to 1091), the ensemble model exhibits the highest mean correlation (0.014), indicating it is the most aligned with actual market movements among all models.

The individual models vary, with "Cyrus20Wisdom" and "Cyrus20FullFeatures" showing higher correlation means than the others, suggesting they have a strong predictive performance. The BMC values seem low, but "Cyrus20Wisdom" stands out with the highest BMC mean (0.008) among the individual models, followed closely by the ensemble (0.011). This suggests that "Cyrus20Wisdom" and the ensemble have unique predictions that contribute positively to the performance when compared to benchmark models. The ensemble model has the highest lpayout Sharpe ratio (2.37), which is significantly higher than any individual model. This suggests that the ensemble model not only predicts well (robust over time in terms of predicted returns) but does so with a favorable risk-return profile. Based on this analysis, the ensemble model appears to be the most effective, in Figure 7, one can see the cumulative metrics of the ensemble model on the testing period.

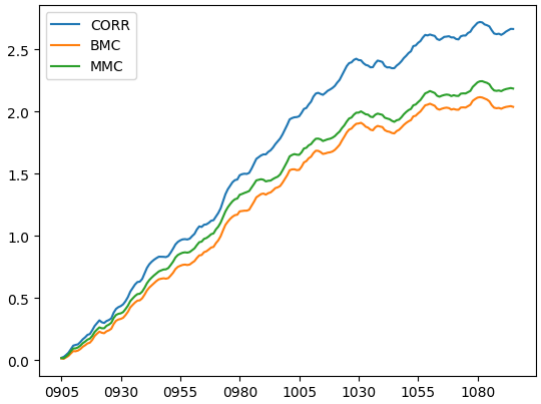


Fig. 7. Ensemble model cumulative metrics

The upward trend in all the three metrics suggests that the ensemble model would have provided 1) positive returns if the predictions were used for live trading, 2) beneficial insights that are not captured by benchmark models, 3) positive contribution even after its predictions have been neutralized to the meta model. The CORR line shows the least volatility and the steadiest growth, which is consistent with the emphasis on CORR being the most robust metric over time. The BMC and MMC lines show more fluctuations, indicating some variability in how the unique contributions of the ensemble model perform over time.

8 Submission

Using Google Cloud VM for automation, the ensemble model, whose predictions not only align well with the actual market but also add value beyond the meta

model and benchmark models, is run daily to provide submissions. The first submission date is February 15, 2024.

References

1. Daniel Poh, Bryan Lim, Stefan Zohren, and Stephen Roberts. 2021. Building Cross-Sectional Systematic Strategies by Learning to Rank.
2. Daniel Poh, Bryan Lim, Stefan Zohren, and Stephen Roberts. 2022. Transfer Ranking in Finance: Applications to Cross-Sectional Momentum with Data Scarcity.
3. Daniel Poh, Bryan Lim, Stefan Zohren, and Stephen Roberts. 2022. Enhancing Cross-Sectional Currency Strategies by Context-Aware Learning to Rank with Self-Attention.
4. Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, Jarosław Bojar. 2020. Context-Aware Learning to Rank with Self-Attention.