Imperial College London
Department of Mathematics

Robust Machine Learning Methods for High-dimensional Datasets with Applications in Genomics and Finance

Ming Hei Wong

Submitted in part fulfilment of the requirements
for the degree of Doctor of Philosophy at
Imperial College London, September 2023

Abstract

This thesis studies different robust machine learning methods towards high-dimensional datasets under concept drifts, using examples from single-cell RNA sequencing and stock trading in quantitative finance.

The diversity of next-generation sequencing methods leads to challenges in integrating datasets prepared by different laboratories. A new batch correction and data integration method based on probabilistic matrix factorisations, integrative Hierarchical Poisson Factorisation (IHPF), is introduced in this thesis. IHPF have robust performance under different noise levels in datasets and provides interpretable latent factors which capture the biological signals in the datasets. The cell and gene factorisation scores obtained are better than its predecessor, Hierarchical Poisson Factorisation (HPF).

Financial datasets are highly non-stationary with a low signal-to-noise ratio, often with regime-dependent behaviours. We propose two new dynamic optimisation techniques, dynamic feature projection and dynamic model stacking, to improve the robustness of model predictions. The techniques can improve the risk-adjusted return of trading portfolios developed based on commonly-used machine learning models, such as Gradient Boosting Decision Trees (GBDT) and Multi-Layer Perceptron Networks (MLP).

We then develop a new incremental learning pipeline to accommodate distribution shifts in data by combining models trained with different parameter settings. The new pipeline dynamically performs hyper-parameter optimisation and model stacking, and model weights are dynamically adapted to different concept drifts types.

Using Jackknife feature group sampling, we created a diverse set of models that can model the level of disagreement between investors in the stock market. Tail risk strategies can be created based on variance of predictions from different models which offers hedging benefits under unfavourable market conditions. A dynamically hedged model can be formulated which achieves similar level of return as the example model provided by Numerai with a significant lower drawdown.

We also studied the robustness of different hyperparameters used to train the XGBoost models. We identify an empirical formula to select the learning rates of GBDT models based on the number of boosting rounds for a given dataset, which allows training of shallow GBDT models with model performances converging towards the generalisation upper bound.

Statement of Originality

This thesis, including both the written materials here and associated program codes, is the product of my own work performed under the supervision of Prof. Mauricio Barahona. Any ideas from the work of other people, published or otherwise, are fully acknowledged and referenced.

Copyright Declaration

Acknowledgements

I would like to thank my supervisor Prof. Mauricio Barahona and members of his research group, for their guidance and support, and for providing me with the opportunity for this PhD. I am grateful to the Wellcome Trust for providing the funding of my research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the thesis we study machine learning methods for high-dimensional tabular datasets under distribution shifts, using datasets from genomics and quantitative finance as examples. The aim is to develop incremental and transfer machine learning methods for datasets under adverse assumptions, such as misspecified models, low signal-to-noise ratios, high data sparsity, as well as data distributions being non-stationary and non-Gaussian.

1.1 Concept Drift and Transfer/Incremental Learning

Transfer/Incremental learning is an important research area for machine learning systems deployed in real life situations. Instead of having a constant dataset for all the training and evaluation, the deployed models are applied on unknown data which can be modelled as a data stream. The data available for training is growing with the time the system is deployed. A key challenge in modelling stream data is concept drift [1, 2, 3] [1] since the relationship between underlying features and targets distributions change over time in unexpected ways.

Different sources of concept drift have been studied [1], such as covariate shift (which deals with changes in the underlying feature distribution between train and test periods), conditional distribution shift (which deals with changes in relationships between features and targets), and the mixture of the two sources of concept drift identified above. Mixtures are the case for a lot of datasets from real-life applications as distribution shifts in data often occur in an entangled way which makes it difficult to separate the effects.

Denote $\mathcal{P}_{train}(X, y)$ to be the train distribution of features X and labels y and $\mathcal{P}_{test}(X, y)$ to be the test distribution of features X and labels y. We have three different sources of concept drift:

- Covariate shift when $\mathcal{P}_{train}(X) \neq \mathcal{P}_{test}(X)$ while $\mathcal{P}_{train}(y|X) = \mathcal{P}_{test}(y|X)$
- Conditional distribution shift when $\mathcal{P}_{train}(y|X) \neq \mathcal{P}_{test}(y|X)$
- Mixture of covariate shift and conditional distribution shift when $\mathcal{P}_{train}(X) \neq \mathcal{P}_{test}(X)$ and $\mathcal{P}_{train}(y|X) \neq \mathcal{P}_{test}(y|X)$

Concept drift can be observed in four different ways as follows. Note that the classification is not meant to be complete as real life datasets can often present complex scenarios that cannot be easily fitted into one of the following ways.

[1] In some literature, concept drift is also defined as distribution shifts

- Sudden Drift: A new concept occurs within a short time
- Gradual Drift: A new concept gradually replaces an old one over a period of time
- Reoccurring concepts: An old concept may reoccur after some time

Framework for handling concept drift To handle concept drift in machine learning, extra steps are added after standard model training and prediction. These steps are concept drift detection, understanding and adaptation. In the following, we follow mostly [2] to give a high level summary on the key research progress on concept drift adaptation.

To detect concept drift, a general framework of four different stages are suggested by [2].

- Data Retrieval: Get data with suitably sized windows from the data stream
- Data Modelling: Perform dimensionality reduction if needed to extract features summarising the retrieved data
- Calculating test statistics: Quantify the severity of drift using different dissimilarity measures
- Conducting hypothesis tests: Use a specific hypothesis test to calculate the statistical significance of concept drift

Different drift detection algorithms have been used, each with its own merits and limitations. The most common one would be error rate-based drift detection methods [2, 4], which focuses on tracking changes in the online prediction loss of the models. A model update will be triggered if the changes in prediction losses are outside the statistical bounds. Data distribution-based algorithms are also used. These algorithms quantify the distance/dissimilarity between the distribution of historical and recent data. More recent methods include multiple hypothesis test algorithms which employs multiple tests in parallel and in a hierarchical manner to reduce false alarms on concept drifts. Drift detection algorithms often can also be used to identify the time when concept drift occurs and their severity, providing an understanding on the cause of the concept drift in the data.

Many strategies have been used to update existing machine learning models to the drift in the data. These strategies can be grouped in three categories, namely model retraining, ensemble retraining and model updating, which can each be tailored to handle different types of concept drift.

Model retraining is the simplest method to adapt to concept drift. A new model is trained with the most recent data to completely replace an old model trained on previous data. Two key decisions have to be made for model retraining, namely the window size of the training set (which can be fixed or dynamically decided based on drift detection algorithms), and retrain frequency (which again can be fixed or dynamically determined). Updating the models too frequently would result in over-fitting towards recent data, whilst updating the models not frequently enough would result in under-fitted models. Similarly, choosing a larger window size of training set can provide more data for model learning, but a smaller window size can adapt to distribution shifts quicker, as older data that are less relevant are removed sooner.

Ensemble retraining extends standard model ensemble techniques such as bagging, boosting and random forests. These methods modify the standard gradient boosting algorithm to adaptive ones, which adds and deletes base learners (decision trees) based on recent performance. Examples include the Adaptive Random Forest algorithm [5] and Random Ensemble Decision Trees [6]. Some ensemble methods employ new voting techniques to handle concept drift. For example, Dynamic Weighted Majority

[7], which down-weights base learners that mis-classify recent data and then trains new base learners to add to the ensemble. There are also methods with more complex voting rules (such as the hierarchical ensemble [8]) and dynamic ensemble sizes [9].

Model updating involves different techniques which modify an existing model to adapt to concept drift instead of completely retraining the model. These methods are often more efficient than retraining. One Examples is the Very Fast Decision Tree Classifier [10], which uses Hoeffding bound to accelerate node splitting, a time consuming step in building decision trees.

A key limitation of the above concept drift adaptation models is that they are developed based on CPU architectures and are not accelerated by GPU's, which is now the standard hardware used in machine learning. Many methods use customised versions of decision trees and gradient boosting algorithms, which can be less optimal than popular packages such as XGBoost [11], LightGBM [12] and CatBoost [13]. Another limitation is that many of these methods introduce ad hoc rules for tree building/removal which do not have a clear justification. It is not certain the default hyper-parameters, for these rules chosen in the methods, would work well in general for datasets not considered by their studies. Most online decision trees algorithms focus only on classification problems and cannot be directly used for regression or ranking problems, which limit their use cases.

Other challenges in incremental learning Here, we discuss some challenges for incremental learning other than above mentioned concept drift, highlighting the differences between incremental learning and standard machine learning.

Look-ahead bias is common for time-series forecast problems and other machine learning problems involving data streams. The temporal ordering of observations in data streams requires researchers to carefully design model training pipelines, so that future information is not leaked into model design and training.

Incremental learning tasks would use datasets that are usually much larger than datasets studied in most bench-marking studies [14, 15, 16]. The computational resources required by some machine learning methods, in particular deep learning, limits the choice of machine learning approaches that can be used. Some research [16] has demonstrated that for larger datasets, Gradient Boosting Decision Trees (GBDT) performed better than 11 neural-network-based approaches and 5 baseline approaches, such as Support Vector Machines (SVM). This further discourages the use of neural-network-based approaches, since these approaches require more than an order of magnitude of training time when compared to GBDT models implemented in XGBoost [16].

1.2 Challenges in transfer learning

In the following, we discuss some challenges in transfer learning [17] and how different approaches can be used to overcome those challenges.

1.2.1 Randomness in model training

A key challenge in machine learning research is understanding the impact of randomness on model training. Randomness in the model learning process results in variation in model performance and it is therefore recommended to report model performances across multiple runs with different random seeds [17].

In deep learning models, sources of randomness include randomisation of weights initialisation, stochasticity from dropout layers, and order of data batches in stochastic gradient descent updates. In

GBDT models, data sampling before growing each tree and feature sampling before growing each tree, layer or node add randomness to the training process. Faster histogram optimised approximations of greedy algorithms used in tree construction also contribute towards randomness of the process.

Lottery Ticket Hypothesis (LTH) [18] suggests deep learning models contain sparse sub-networks which can be trained in isolation to achieve comparable performances of the full network. When these sparse sub-networks are reinitialised with parameters drawn from different random seeds, these networks can no longer be trained to match the performances of the original network.

The finding is problematic since it suggests the ingenuity of neural network architectures, exemplified by sparse sub-networks in the above are highly dependent on having particular sets of initial weights, which are governed by the random seeds. Indeed, researchers failed to replicate the success of novel neural network architectures, such as TabNet [19], in subsequent bench-marking studies [16, 14].

Variances due to Data Sampling The stochastic nature of data sampling during cross-validations may also have an impact on the model performances being reported. Because of this, some researchers report results over different cross-validations splits when feasible. For incremental learning problems on data streams, cross-validation is restricted by the fact that training data must come before validation data and validation data must come before test data. However, researchers now have to decide the size of the training set and the frequency of model retraining/updates. It is also common to down-sample time-series, mostly at regular intervals to reduce model training time. Again, various choices in allocating training and validation data by randomness or convenience of the researcher all contributed to variation in model performances.

Randomness in Data Augmentation In deep learning, it is common to apply various data augmentations before model training [20, 21]. For example it has been shown that image classification using deep learning models are not robust to small perturbations in data [22], due to over-fitting. Small differences in data preprocessing can impact model performance even when the rest of network architecture and training process are unchanged. Some studies do not report all the details for data preprocessing [17], such that other researchers cannot fully replicate their results.

Default values of machine learning algorithms Deep learning models in general have more hyper-parameters than GBDT models and model performance is sensitive to the choice of these hyper-parameters. Default values are commonly used in deep learning studies, since it is impossible to iterate over all hyper-parameters combinations. The setting of default values by researchers can also be considered as a source of randomness as researchers pick a particular value, either by intention or without much consideration, from a range of candidate values, which is not documented in the studies. It is not certain as to whether these default values are chosen to inflate the favoured model performances or are indeed good design choices that works well in a wide of situations.

From both empirical evidence and a theoretical perspective, deep learning models have more sources of variation than other simpler machine learning models such as decision trees and ridge regression. The reason is because deep learning models have many more parameters that need random initialisation at the start of training process, and rely on stochastic gradient descent for each update.

1.2.2 Hyper-parameter optimisation

Existing literature on bench-marking machine learning models often provides contradictory results [14, 23, 15]. One possible reason is due to the differences in hyper-parameter optimisation in various studies. There are many ways for researchers to introduce bias into hyper-parameter optimisation process, from the design of the search grid of hyper-parameters for the machine learning models to the choice of tuning algorithms used in hyper-parameter optimisation.

Selective tuning of hyper-parameters

A common issue in hyper-parameter optimisation is the selection tuning of algorithms [24, 17], where researchers do not fairly give each machine learning algorithms comparable amount of resources in hyper-parameters tuning. This is further complicated by the fact that different machine learning algorithms have different resource usage.

There are two common ways for researchers to allocate computational resources during bench-marking of algorithms. The first is to allocate the same number of searches for each algorithm, the second is to allocate the same amount of time and hardware for each algorithm. Each method has its own merits and limitations.

The constant search number removes the effects of implementation efficiency and computation hardware and allow the bench-marking process to focus more on the theoretical aspects of the algorithms. However, it ignores the size differences of hyper-parameter spaces between different algorithms. Models with more hyper-parameters would in general require a larger number of searches to tune.

The second method is more reflective of real world practicalities, since hyper-parameter tuning for machine learning systems deployed in real life applications are often limited by the computational resources available. Therefore, this method allows us to estimate the optimal performance we can get under different resources constraints which is important for making business decisions. However, results from this method depend the hardware and software architecture used by the researchers and so may not be replicable by others.

Hyper-parameter optimisation algorithms

Some research [25] suggests the choice of hyper-parameter optimisation algorithms (such as random search, grid search and Bayesian methods) can cause variation that significantly impact the optimal hyper-parameters found. As researchers rarely run multiple random seeds for a single hyper-parameter combination during optimisation, due to computational constraints, the joint interaction between randomness of training a single model and hyper-parameter optimisation is not taken into account. Randomness in the hyper-parameter optimisation process can cause the lack of robustness in model training. For example, the random seeds for random search and Bayesian methods needs to be fixed so that consistent set of hyper-parameters are found. If these sources of randomness are not fixed, it is possible are different sets of hyper-parameters are found, and it is not certain whether model performances are comparable in these cases.

Joint effects between hyper-parameters

Joint effects between hyper-parameters, for example, the number of boosting rounds and learning in GBDT models, the number of training epochs and the learning rate in deep learning models make the optimisation problem more complex. Join effects prevents the use of greedy approach to optimise each

variable independently. It is well known that for multi-variate optimisation problems, the solution found by greedy approach does not always corresponds to the global extremum. The solution found by greedy approaches are path dependent, depending on the order of variable optimisation.

Indeed, all optimisation algorithms suffers from the curse of dimensionality [26]. As deep learning models have more hyper-parameters and thus a larger search grid in general, this means optimising the hyper-parameters for deep learning is more difficult than other machine learning models, such as random forests.

1.3 Project Overview

The thesis consists of three sub-projects working with different types of tabular datasets. We first give a brief introduction to the datasets used in the thesis and introduce the concept of robust machine learning. We then describe the three projects that encompass the main content of the thesis. In the first project, we introduce a new matrix factorisation algorithm for data integration in scRNA-seq datasets. In the second project, we conducted a robust study of dynamic machine learning methods for temporal tabular data with regime changes, using the dataset from Numerai [27]. Numerai is an ongoing data science competition based on financial data. In the last project, we developed a new deep incremental learning framework for non-stationary datasets and study the effects of data sampling and model complexity on the variation of model performances.

1.3.1 Temporal and Batch Tabular Datasets

Tabular data is one of the most widely studied data types in machine learning. A tabular dataset X is defined as a two-dimensional tensor of size (N, M), where N is the number of observations and M is the number of features.

Tabular datasets are often organised in batches. In particular, a batch tabular dataset is defined as a collection of K tabular datasets, where each dataset k $(1 \leq k \leq K)$ has size (N_k, M) where N_k is the number of entities in batch k and M is the number of features. If the dataset is collected over time, then the dataset is called a *temporal tabular dataset* (and the batches are called timesteps). Each row in the batch/temporal tabular dataset is uniquely identified by the (batch/timestep, entity) pair.

There are two different categories of temporal tabular datasets. Firstly, when the time and entity mapping is known, the temporal tabular dataset is called a *panel* dataset which can be organised in both cross-sectional and longitudinal formats. Cross-sectional formats represent the snapshot of the dataset at a single time-step, which corresponds to each dataset k, as defined above. Longitudinal formats represent grouping observations from a single entity over a shorter time frame to form a multi-variate time series over the features. Under the assumption where the entities do not change over time in data collection, N multi-variate time-series of length K with M features can be obtained.

Secondly, when the time and entity mapping is unknown, the temporal tabular dataset cannot be organised in longitudinal formats directly. In this case, the number of entities at each time step does not have to be equal. Time Series methods cannot be applied directly. In bioinformatics and healthcare studies, the second category of temporal tabular datasets is more common. For example, participants might drop out of clinical trials. Challenges in survival analysis such as censoring, truncation and missing data are common in healthcare datasets. Privacy considerations will also prevent the identification of (time-step, entity) pairs in some datasets. Therefore, temporal relationships in the dataset need to be modelled indirectly using techniques from transfer and incremental learning.

Choice of Batch Tabular Datasets Benchmark datasets commonly used in scRNA-seq analysis are used to study dimensionality reduction methods, which are robust and interpretable. A detailed description of the datasets is given in Chapter 2.

Choice of Temporal Tabular Datasets To study temporal tabular datasets with distributional shifts, many openly available datasets are considered. The Numerai dataset [27, 28] is chosen because the dataset meets the following criteria on dataset quality, in particular the criteria that the dataset is incrementally updated to provide double-blind out-of-sample data, which are explained below.

- Data Quality: The dataset is generated following a robust process with quality control
- Documentation: Clear documentation is provided for the dataset
- Standardised: The dataset should be pre-processed by standard methods and in a numerical format.
- Incrementally updated: Live (unseen) data should be updated incrementally so that out-of-sample performances can be computed without any look-ahead bias.

A key issue of using *static* datasets from Kaggle [2] or UCI Machine Learning [3] to study *incremental* learning models is that no double-blind out-of-sample can be generated. In this context, double-blind refers to both researchers(data scientists) and subjects(data providers) not being able to know the data in advance. Double-blind datasets remove human bias from the model evaluation process and thus offer a robust evaluation of the merit of different machine learning modelling approaches.

Numerai [27] provides obfuscated datasets consisting of standardised numerical features and targets for a temporal ranking task. Creating such datasets would be prohibitively expensive for researchers, as it uses very high quality data from commercial financial databases. Due to the commercial sensitive nature of dataset, Numerai cannot provide detailed explanation on the data generation process. However, the quality of data is being assured indirectly by the fact it is used to create trading signals for a hedge fund managing over $100M$ USD.

1.3.2 Designing Robust Machine Learning Models

Many high-risk applications of machine learning, such as self-driving cars or trading, are not concerned with just average performances(such as classification accuracy, and profit), but also worse-case performances (such as risks to pedestrians, estimated maximum loss). There are multiple objectives in the optimisation process that may be opposed to each other. For example, by taking more risky investment decisions, a trading model may make more money on average, but the estimated maximum loss will also likely increase. This is very different from most traditional machine learning applications, where models are often evaluated based on a single metric (such as Mean-Square Error Loss, Accuracy), which ignores the tail behaviour of predictions. For applications that are non-ergodic, such as trading, the auto-correlation and other temporal measures of model performances also needs to be taken into account.

Existing work on robust machine learning [29, 30] focuses on how to compare different machine learning approaches in a robust way by removing different sources of randomness. Most incremental learning problems are non-stationary by nature, and as suggested by the No Free Lunch Theorem [31], it is unlikely that there exists a single method that consistently outperforms all others. As a result, it

[2] https://www.kaggle.com
[3] https://archive.ics.uci.edu/

is more practical to create model ensembles, which have a more stable performance over different data regimes, rather than finding a particular machine learning algorithm that outperforms by chance.

When models have similar performances, we then select models based on their interpretability and the complexity of algorithms. Interpretable models are preferred to black-box approaches such as deep learning.

1.3.3 Dimensionality Reduction and Data Integration for scRNA-seq data based on Integrative Hierarchical Poisson Factorisation

In chapter 2, Integrative Hierarchical Poisson Factorisation (IHPF), an extension of Hierarchical Poisson Factorisation for data integration in scRNA-seq analysis is presented. Data integration can be considered as a specific example of transfer learning, where the aim is to reduce distributional differences between different batches of genomics data. The bench-marking of different data integration methods on three carefully designed scenarios is performed, in order to capture a wide range of possible use cases. Models are evaluated on different measures of both robustness and interpretability. IHPF is shown to be a robust and interpretable method to generate latent factors for scRNA-seq, which is useful for understanding the underlying biological phenomena and is better than standard methods, such as Principle Component Analysis (PCA) and recent deep learning based methods such as single-cell variational inference (scVI).

1.3.4 Online Learning Techniques for Prediction of Temporal Tabular Datasets with Regime Changes

The application of deep learning to non-stationary temporal datasets can lead to overfitted models that underperform under regime changes. In chapter 3, we propose a modular machine learning pipeline for ranking predictions on temporal panel datasets which is robust under regime changes. The modularity of the pipeline allows the use of different models, including Gradient Boosting Decision Trees (GBDTs) and Neural Networks, with and without feature engineering. We evaluate our framework on financial data for stock portfolio prediction, and find that GBDT models with dropout display high performance, robustness and generalisability with reduced complexity and computational cost. We then demonstrate how online learning techniques, which require no retraining of models, can be used post-prediction to enhance the results. First, we show that dynamic feature projection improves robustness by reducing drawdown in regime changes. Second, we demonstrate that dynamical model ensembling based on selection of models with good recent performance leads to improved Sharpe and Calmar ratios of out-of-sample predictions. We also evaluate the robustness of our pipeline across different data splits and random seeds with good reproducibility.

1.3.5 Deep Incremental Learning Models for Temporal Tabular Datasets with Distribution Shifts

In chapter 4, we present a robust deep incremental learning framework for regression-based ranking tasks on financial temporal tabular datasets from Numerai 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. Various strategies of model ensemble over different training settings, such as sizes of training set, learning rates can be used to enhance the robustness of

predictions. Model ensembles trained with different targets can create strategies with better risk profile. Training different models using Jackknife sampling of feature groups can create diverse models that are good approximation as the diverse behaviour of investors in the market. Tail risk strategy can be created based on variance of models within the ensemble, as uncertainty and disagreement between investors are indicators of regime changes in data. Hedging the baseline ensemble prediction based on simple averages with the tail risk strategy can create strategies better than benchmark model provided by Numerai. Detailed analysis on the robustness on model hyperparameters and data sampling schemes are performed also.

Chapter 2

Dimensionality reduction and data integration for scRNA-seq data based on integrative hierarchical Poisson factorisation

2.1 Introduction

Recent advances in sequencing technologies make it possible to profile biological processes at single-cell resolution, allowing the characterisation of the cell-to-cell variability that underpins numerous phenomena in biology and medicine. Single-cell RNA sequencing (scRNA-seq) has a wide range of applications measuring cell heterogeneity, from cancerous tumours to antibiotic resistance in bacterial populations [32, 33]. In scRNA-seq data sets, each sample is a cell described by a high-dimensional feature vector containing the transcription counts of tens of thousands of genes. Hence the *sample space* correspond to cells, and the *feature space* refers to genes. The feature vectors are discrete, noisy and sparse, with many zeros and low counts.

Due to the large number of features (i.e., genes), dimensionality reduction is a key ingredient in the analysis workflow of scRNA-seq data sets. Hence many computational pipelines are designed to extract a reduced set of informative and interpretable *latent factors* that capture the distinguishing characteristics of the cell types in the data set. However, the sparse and discrete nature of scRNA-seq data poses challenges to classic methods of dimensionality reduction, e.g., Principal Component Analysis (PCA) or Non-negative Matrix Factorisation (NMF). An additional challenge is posed by data integration. It is common to integrate data sets sequenced with different technologies and across different laboratories to validate hypotheses against previous findings, or to uncover rare biological signals that are difficult to observe in individual experiments. For example, researchers might combine a newly sequenced data set with existing reference data sets to identify cell types. Under such data integration scenarios, biologically meaningful signals must be separated from technical effects, such as differences in sequencing depth, quality control, or population effects (e.g., demographics of patients providing the samples) [34]. Such batch effects are often non-linear, and hence problematic for batch correction methods based on, e.g., PCA [35].

There are two major data integration tasks in single-cell genomics. One type of task is multi-omics integration, which aims to combine different modalities of data (e.g., transcriptomics and proteomics) through *a shared cell space*, as done by tools such as MOFA+ [36] and DIABLO [37]. A second integration task aims to combine data sets of the same modality but collected from different experiments, and correct batch differences *on a shared gene space*. Examples of these methods include scVI [38] and Scanorama [39]. Here, we focus on this latter task.

Several tools have been recently deployed for batch correction of scRNA-seq data ranging from traditional matrix factorisation, such as PCA or NMF, to deep learning methods, such as variational autoencoders. For a survey, see [35, 40]. A common characteristic of these methods is that they involve dimension reduction of the data followed by graph-based clustering using, e.g., k-nearest neighbour graphs. Batch correction is applied on the distance between cluster centroids under a suitable metric. However, these methods typically require long computational times, thus limiting their use for large-scale single-cell studies [41], and can lack robustness, with performance highly dependent on data pre-processing and gene filtering, on specific choices of hyper-parameters, and on fine-tuning for particular sequencing technologies [40].

An additional, desirable requirement in dimensionality reduction of scRNA-seq data sets is that latent factors are interpretable in both the cell and gene spaces, so that they can be used to generate biologically meaningful data-driven hypotheses linking gene expression and cell phenotypes. However, many of the current methods (e.g., PCA) lack interpretability due to the signed nature and lack of sparsity of the latent factor scores in the gene or cell spaces. Several methods have been proposed to improve interpretability. In particular, NMF ensures latent factors that learn a part-based structure, with non-negative scores, but with no guarantee of sparsity. Lack of interpretability is compounded, and especially relevant, when dealing with data integration of cell types across technical batches.

In this chapter, we focus on data integration and dimensionality reduction of scRNA-seq data sets comprised of several batches, and consider this problem in the context of matrix factorisation methods that induce robust and interpretable latent factors to simplify downstream analyses. To do so, we introduce Integrative Hierarchical Poisson Factorisation (IHPF), an extension to HPF that allows data integration and provides a robust, flexible and interpretable method for the analysis of multi-batch scRNA-seq data sets under different data integration scenarios.

The chapter is organised as follows. Section 2.2 provides background on existing factorisation methods for dimensionality reduction pertaining to scRNA-seq analyses. Section 2.3 describes our algorithm for Integrative Hierarchical Poisson Factorisation (IHPF), the data sets, our numerical experiments, and we demonstrate how IHPF can be used to carry out dimensionality reduction under different data integration scenarios. In Section 2.4, we show that the latent factors obtained by IHPF provide a good basis for cell clustering and visualisation, and are also sparse and interpretable due to their dual block-structure in both cell and gene spaces.

2.2 Background

2.2.1 Matrix factorisation and dimensionality reduction

Consider a data set consisting of N samples, each of them described by $M \gg 1$ features. In the case of scRNA-seq, the N samples are cells of different types, and each of the M features is the transcription count of a particular gene. The data is compiled into a data matrix X of dimensions $N \times M$.

Matrix factorisation methods attempt to find $\widehat{X}$, an approximation of the data matrix X in terms of

two matrices W and V with reduced dimension (rank) $K \ll M$ such that their product is close to the original matrix:

$$X_{N\times M} \approx W_{N\times K} V_{K\times M} =: \widehat{X}. \tag{2.1}$$

Here, the matrix W contains the cell scores and the matrix V contains the gene scores with respect to K *latent factors*. The latent factors thus recapitulate (approximately) the information contained in the M features, and each sample (cell) can be described by K coordinates.

PCA and NMF are two classic methods that produce such a matrix factorisation, differing in how the approximation (2.5) is computed. PCA minimises the Frobenius norm

$$\min_{W,V} \|X - WV\|_F, \tag{2.2}$$

and it is well known that PCA approximations are obtained through the singular value decomposition (SVD) of the matrix X, i.e., the K factors are obtained from the top K singular vectors of X ordered in decreasing order of their singular values [42]. PCA-based methods have been widely used with scRNA-seq data sets, typically after applying a log-transformation to the raw data counts. However, the fact that the cell and gene scores (W and V) are signed and not sparse, together with the log-transformation, reduces the interpretability of the latent space.

The data matrix in scRNA-seq is non-negative element-wise ($X \geq 0$); hence a factorisation that captures this property is desirable. NMF proposes such a factorisation: it minimises the Frobenius norm but imposing non-negativity (element-wise) of the factorisation matrices [43]:

$$\min_{W,V} \|X - WV\|_F \quad \text{subject to} \quad W \geq 0,\, V \geq 0. \tag{2.3}$$

NMF learns part-based representations, which represent positive contributions of the latent factors to the samples [43], yet with no guarantee of sparsity.

2.2.2 Probabilistic viewpoint of factorisation

Dimensionality reduction (and its associated matrix factorisations) also has probabilistic interpretations, specifically in terms of probabilistic factor analysis (FA) and hierarchical models.

From a probabilistic perspective, it has been shown [44] that PCA can be understood as a generative model where the samples are noisy observations of a combination of latent factors that are multivariate Gaussian [45, 46]:

$$\begin{aligned} X_{N\times M} &\sim \text{Gaussian}(WV, \sigma^2 \mathbb{I}) \\ W_{N\times K} &\sim \text{Gaussian}(0, \mathbb{I}) \\ V_{K\times M} &\sim \text{Gaussian}(0, \mathbb{I}) \\ \sigma &\sim \text{log-Normal}(0, 1). \end{aligned}$$

NMF also has a probabilistic interpretation related to Poisson factor analysis (PFA) [47]. It can be shown that under NMF, X is assumed to be a matrix of Poisson random variables with rates given by a

factorisation into two matrices:

$$
\begin{aligned}
X_{N\times M} &\sim \text{Poisson}(\Theta\,\Delta)\\
\Theta_{N\times K} &\sim \text{Gamma}(1,\infty)\\
\Delta_{K\times M} &\sim \text{Gamma}(1,\infty).
\end{aligned}
$$

Here, Θ and Δ correspond, respectively, to cell and gene scores onto the K latent factors Θ and are Gamma distributed random variables with shape parameter 1 and infinite rate. Note that Poisson-Gamma random variables are related to the Negative Binomial distribution, which is widely used to model scRNA-seq counts.

Recently, this probabilistic interpretation inspired the Hierarchical Poisson Factorisation (HPF) [48], which uses a hierarchical Gamma prior:

$$
\begin{aligned}
X_{N\times M} &\sim \text{Poisson}\,(\Theta\,\Delta)\\
\Theta_{N\times K} &\sim \text{Gamma}(a,\Xi)\\
\Xi_{N\times K} &\sim \text{Gamma}(a',b')\\
\Delta_{K\times M} &\sim \text{Gamma}(c,\mathrm{Z})\\
\mathrm{Z}_{K\times M} &\sim \text{Gamma}(c',d')
\end{aligned}
$$

where the cell scores Θ follow a Gamma distribution with shape parameter a and a Gamma distributed rate Ξ. Similarly, the gene scores Δ are Gamma distributed with shape parameter c and a Gamma distributed rate Z.

HPF has been shown to perform better than NMF when applied to sparse data sets since the hierarchical Gamma prior is more flexible in modelling zero-inflated data counts [48].

2.2.3 Data integration

The problem of integrating data sets obtained with different technologies under diverse experimental settings poses additional challenges, as one is usually interested in capturing biologically meaningful signals in feature (gene) space under technical batch effects in sample (cell) space.

The classic matrix factorisation methods introduced in Section 2.2.1 have been extended for data integration. Traditionally, PCA has been used for batch correction in bulk RNA transcriptomics data, but it does not perform satisfactorily on scRNA-seq data due to the high sparsity of single cell data [33]. To circumvent these limitations, Scanorama [39] and BBKNN [49] apply graph-based algorithms on the principal components extracted by PCA to remove batch effects and improve alignment between cell types. BBKNN uses a k-nearest neighbour algorithm to build a graph to capture cell-to-cell similarity, attempting to balance the contribution from each batch. Scanorama identifies mutual nearest neighbours between batches and then finds the optimal order of batch alignment for panorama stitching, followed by a non-linear transformation to create the integrated latent space and batch-corrected gene expression matrices.

Integrative NMF (INMF) extends NMF to harmonise data sets that share their gene space [50]. For S batches with non-negative data matrices $X_s \in \mathbb{R}_+^{N_s\times M}$, $s = 1,\ldots,S$, with $N = \sum_{s=1}^{S} N_s$, INMF finds a description in terms of K latent factors by obtaining non-negative matrices $W \in \mathbb{R}_+^{N\times K}$, $V_s \in \mathbb{R}_+^{K\times M}$, $s = 1,\ldots,S$, and $B \in \mathbb{R}_+^{K\times M}$ containing, respectively, cell scores, gene scores specific to each data set, and

gene scores common to all data sets. The INMF factorisation minimises the loss function:

$$\sum_{s=1}^{S} \|X_s - W(\mathbf{1}_s, :)(V_s + B)\|_F + \frac{1}{\alpha} \sum_{s=1}^{S} \|W(\mathbf{1}_s, :)V_s\|_F \ . \tag{2.4}$$

where $\mathbf{1}_s$ is the indicator function for cells in batch s and $W(\mathbf{1}_s, :) \in \mathbb{R}_+^{N_s \times K}$ is the submatrix containing the cell scores for batch s. The hyper-parameter α, denoted as the *noise ratio*, regularises the magnitude of the components of the batches and assigns the variability in the data between cell types and batches.

Recently, deep learning has been used to perform dimensionality reduction and batch correction at the same time. scVI [38] uses a Variational Auto-Encoder (VAE) to learn a latent space representation of cells, modelling both library size and batch effects using a zero-inflated negative binomial distribution for gene expression counts. A significant drawback is that the latent space representation is not interpretable.

2.3 Methods

2.3.1 Integrative Hierarchical Poisson Factorisation (IHPF)

It has been shown that the performance of HPF can suffer when integrating data sets of different provenance (e.g., batches or other sample effects); hence various *ad hoc* methods (e.g., clustering of biologically related latent factors) have been applied to ameliorate batch effects [51].

Here we introduce an algorithm for *Integrative Hierarchical Poisson Factorisation* (IHPF), which extends the applicability of HPF to allow for systematic data integration, as follows.

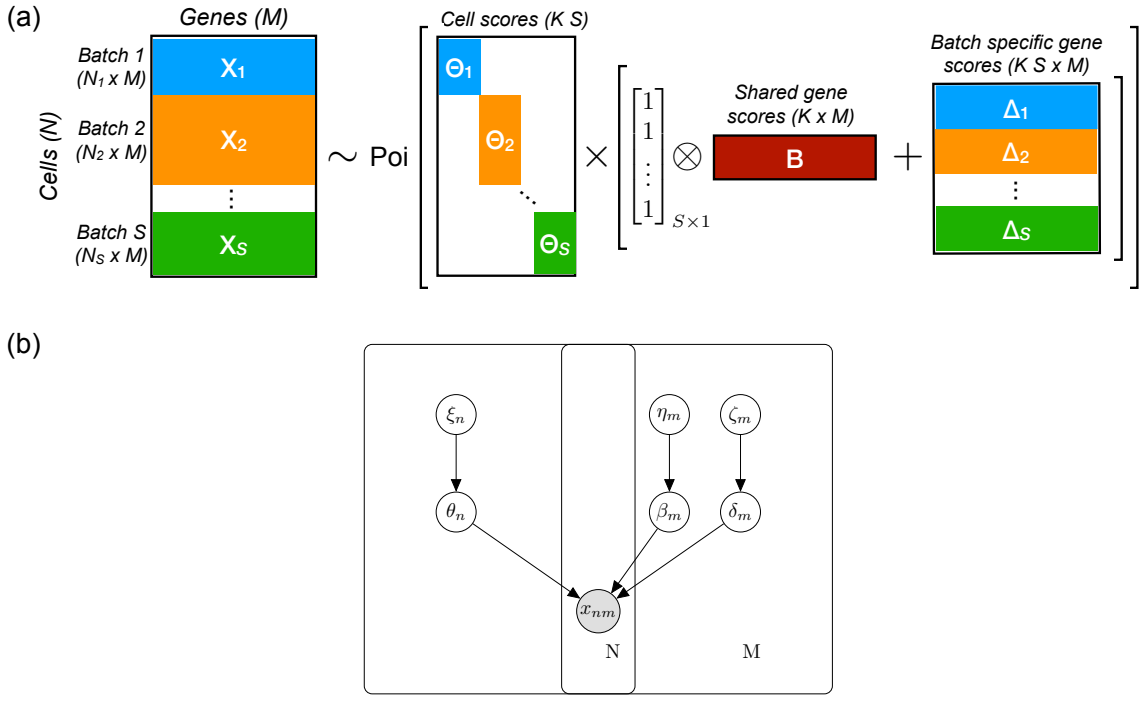


Figure 2.1: Two representations of the IHPF model. (a) IHPF can be viewed as a probabilistic matrix factorisation where the count matrix X, formed by the X_i of the S batches, is modelled as a Poisson distribution with weights given by a matrix factorisation with cell scores Θ_i together with batch-specific gene scores Δ_i and common gene scores B; (b) Graphical representation of the hierarchical IHPF model with N cells and M genes.

Given data matrices $X_s \in \mathbb{R}_+^{N_s \times M}$, with $s = 1, \ldots, S$ indexing S batches, IHPF assumes a hierarchical Gamma prior on the cell scores Θ and it decomposes the gene scores into components $\Delta^{(s)}$, accounting for batch-specific effects, and a component B, accounting for shared gene effects. Both $\Delta^{(s)}$ and B are also described by a hierarchical Gamma prior. Then the IHPF model is given by:

$$
\begin{aligned}
X_{N_s \times M}^{(s)} &\sim \text{Poisson}\left(\Theta(\mathbf{1}_s, :)\left(\Delta^{(s)} + \mathrm{B}\right)\right), \; s = 1, \ldots, S \\
\Theta_{N \times K} &\sim \text{Gamma}(a, \Xi) \\
\Xi_{N \times K} &\sim \text{Gamma}(a', b') \\
\Delta_{K \times M}^{(s)} &\sim \text{Gamma}(c, \mathrm{Z}^{(s)}), \quad s = 1, \ldots, S \\
\mathrm{Z}_{K \times M}^{(s)} &\sim \text{Gamma}(c', d'), \quad s = 1, \ldots, S \\
\mathrm{B}_{K \times M} &\sim \text{Gamma}(c, \mathrm{H}) \\
\mathrm{H}_{K \times M} &\sim \text{Gamma}(c', d'),
\end{aligned}
$$

where, again, $\Theta(\mathbf{1}_s, :) \in \mathbb{R}_+^{N_s \times K}$ is the submatrix containing the cell scores of cells in batch s. The pseudo-code for the IHPF generative model is presented in Algorithm 7 and the IHPF algorithm is summarised in Figure 2.1.

Algorithm 1: IHPF model for single-cell data.

Set hyper-parameters a, a', b', c, c', d'
for each cell i **do**
  Sample rate of cell score: $\xi_i \sim \text{Gamma}(a', b')$
  for each factor k **do**
    Sample cell score: $\theta_{ik} \sim \text{Gamma}(a, \xi_i)$
  end for
end for
for each gene g and batch s **do**
  Sample rate of shared gene score:
    $\eta_g \sim \text{Gamma}(c', d')$
  Sample rate of batch-specific gene score:
    $\zeta_{gs} \sim \text{Gamma}(c', d')$
  for each factor k **do**
    Sample shared gene score:
      $\beta_{gk} \sim \text{Gamma}(c, \eta_g)$
    Sample batch-specific gene score:
      $\delta_{gsk} \sim \text{Gamma}(c, \zeta_{gs})$
  end for
end for
for cell i and gene g **do**
  Denote batch label of cell i to be s
  Sample observed expression level:
    $x_{ig} \sim \text{Poisson}\left(\sum_{k=1}^{K} \theta_{ik}(\beta_{gk} + \delta_{gsk})\right)$
end for

We use mean-field variational inference [52] to infer the IHPF model. Algorithm 8 describes the coordinate ascent algorithm [48] used to infer the shape and rate parameters of the six Gamma posteriors

for gene and cell scores in Algorithm 7:

$$\xi_i|X,\ \theta_{ik}|X,\ \eta_g|X,\ \zeta_{gs}|X,\ \beta_{gk}|X,\ \delta_{gsk}|X,$$

where each is Gamma distributed and the shape and rate parameters are defined accordingly for each posterior, e.g.,

$$\xi_i|X \sim \text{Gamma}(\xi_i^{shp}, \xi_i^{rte}).$$

Algorithm 2: Coordinate Ascent Algorithm for likelihood inference in IHPF.

Set the number of factors K, noise ratio α, and hyper-parameters a, a', b', c, c', d'.

Initialise allocations to gene latent factors with Dirichlet prior:

$(\phi_{ig1}, \ldots, \phi_{igK}, \omega_{ig1}, \ldots, \omega_{igK}) \sim \mathrm{Dir}(1, \ldots, 1, \alpha, \ldots, \alpha)$

repeat

for each gene g **do**

Compute shapes and rates of Gamma posterior of shared gene scores:

$$\beta_{gk}^{shp} = c + \sum_i x_{ig}\phi_{igk}$$
$$\beta_{gk}^{rte} = \frac{\eta_g^{shp}}{\eta_g^{rte}} + \sum_i \frac{\theta_{ik}^{shp}}{\theta_{ik}^{rte}}$$
$$\eta_g^{shp} = c' + Kc$$
$$\eta_g^{rte} = d' + \sum_k \frac{\beta_{gk}^{shp}}{\beta_{gk}^{rte}}$$

for each batch s **do**

Compute shapes and rates of Gamma posterior of batch-specific gene scores:

$$\delta_{gsk}^{shp} = c + \sum_{i \in s} x_{ig}\omega_{igk}$$
$$\delta_{gsk}^{rte} = \frac{\zeta_g^{shp}}{\zeta_g^{rte}} + \sum_{i \in s} \frac{\theta_{ik}^{shp}}{\theta_{ik}^{rte}}$$
$$\zeta_{gs}^{shp} = c' + Kc$$
$$\zeta_{gs}^{rte} = d' + \sum_k \frac{\delta_{gsk}^{shp}}{\delta_{gsk}^{rte}}$$

end for

end for

for each cell i **do**

Denote batch label of cell i to be s

Compute shapes and rates of Gamma posterior of cell scores:

$$\theta_{ik}^{shp} = a + \sum_g x_{ig}(\phi_{igk} + \omega_{igk})$$
$$\theta_{ik}^{rte} = \frac{\xi_i^{shp}}{\xi_i^{rte}} + \sum_g \left(\frac{\beta_{gk}^{shp}}{\beta_{gk}^{rte}} + \frac{\delta_{gsk}^{shp}}{\delta_{gsk}^{rte}} \right)$$
$$\xi_i^{shp} = a' + Ka$$
$$\xi_i^{rte} = b' + \sum_k \frac{\theta_{ik}^{shp}}{\theta_{ik}^{rte}}$$

end for

for each cell i and gene g such that $x_{ig} > 0$ **do**

Use the digamma function $\Psi(\cdot)$ to compute the multinomial distributions:

$$\phi_{igk} \propto \exp\left(\Psi(\theta_{ik}^{shp}) - \log\theta_{ik}^{rte} + \Psi(\beta_{gk}^{shp}) - \log\beta_{gk}^{rte}\right)$$
$$\omega_{igk} \propto \exp\left(\Psi(\theta_{ik}^{shp}) - \log\theta_{ik}^{rte} + \Psi(\delta_{gsk}^{shp}) - \log\delta_{gsk}^{rte}\right)$$

end for

until convergence

The multinomial distributions used in Algorithm 8 for the allocation of gene and cell counts to each latent factor are initialised with a Dirichlet prior with weights given by the noise ratio $\alpha \in [0, 1]$. This hyper-parameter represents the relative importance of shared and batch specific gene latent factors, i.e., how much variance in the data is to be attributed to technical noise between data sets. Taking the limit $\alpha \to 0$ recovers the original HPF.

2.3.2 Data sets and data integration scenarios

We have collated from the published literature 16 scRNA-seq data sets of varying sizes and scope, and collected with different sequencing technologies (Table 2.1). Quality control is applied on the data sets by keeping only cells with at least 600 transcripts.

Table 2.1: Data sets used to benchmark various methods for data integration. Data sets are combined to produce integrated data sets to conform a specific scenario reflecting different assumptions on the biological relationship between data sets.

ID	Description	Technology	Cells (N)	Genes (M)	Sequecing Depth	Scenario	Reference
1	Mouse Jurkat cells	10X Genomics	2885	32378	3060	A	Zheng 1[53]
2	Mouse 293t cells	10X Genomics	3257	32378	3168	A	Zheng 2[53]
3	Mouse 50% Jurkat/50% 293t cells	10X Genomics	3388	32378	3249	A	Zheng 3[53]
4	Human pancreas	InDrop	8569	20126	2622	B	Baron[54]
5	Human pancreas	Cel-seq2	2449	19140	3908	B	Muraro[55]
6	Human pancreas	Fluidigm C1	638	25463	3294	B	Lawlor [56]
7	Human pancreas	Smart-seq2	2989	26179	3342	B	Segerstolpe[57]
8	Human pancreas	Celseq	1276	20148	3196	B	Grün[58]
9	Human CD19+ B cells	10X Genomics	2261	32378	1598	C	Zheng 5[53]
10	Human CD14+ monocytes	10X Genomics	295	32378	1824	C	Zheng 6[53]
11	Human CD4+ helper T cells	10X Genomics	3713	32378	1581	C	Zheng 7[53]
12	Human CD56+ natural killer cells	10X Genomics	6657	32378	1714	C	Zheng 8[53]
13	Human CD8+ cytotoxic T cells	10X Genomics	3990	32378	1509	C	Zheng 9[53]
14	Human CD4+CD45RO+ memory T cells	10X Genomics	3628	32378	1543	C	Zheng 10 [53]
15	Human CD4+CD25+ regulatory T cells	10X Genomics	3365	32378	1609	C	Zheng 11[53]
16	Human PBMC	10X Genomics	2293	32378	1743	C	Zheng 4 [53]

We group data sets in Table 2.1 to conform three biological scenarios:

- *Scenario A:* a collection of data sets sequenced with the same technology and containing different abundance of the same cell types; here marker genes can easily tell the difference between cell types.
- *Scenario B:* a collection of data sets sequenced with different technologies and containing several cell types;
- *Scenario C:* a collection of data sets sequenced with the same technology in which there is little overlap in cell types between batches.

Table 2.2 summarises the three data scenarios. In our case, Scenario A consists of 3 batches of mouse cells (ID 1-3) sequenced with the 10X Genomics technology and comprising two cell types (293t and jurkat) mixed in different proportions. Scenario B consists of 5 batches of human pancreas cells (ID 4-8) of 10 different types obtained with different sequencing technologies. Scenario C consists of 8 batches (ID 9-16) sequenced with the same technology (10X Genomics) and comprising 6 different cell types highly concentrated in particular batches.

These three scenarios reflect experimental situations with different alignment between cell types and batch labels. In Scenario A, batches and cell types have some alignment and, importantly, there is good overlap of cell types across batches. In Scenario B, the situation is complex, with many batches and cell types and with differing overlap of cell types across batches. In Scenario C, batches and cell types are

Table 2.2: Scenarios for data integration constructed from individual data sets from Table 2.1.

	Scenario A (ID 1-3)	Scenario B (ID 4-8)	Scenario C (ID 9-16)
Total number of cells, N	9530	15921	26202
Number of common genes, M	32643	15369	32643
Number of batches, S	3	5	8
Number of cell types, C	2	10	6
AMI cell types vs. batches	*0.498*	*0.032*	*0.842*

highly aligned with little overlap of cell types across batches. To measure the alignment between cell types and batches, we use the Adjusted Mutual Information (AMI), as shown in Table 2.2.

Scenario A is an 'easy' scenario with few cell types well sampled across batches; hence good results from different methods are expected. Scenarios B and C represent two extremes of data integration in terms of cell types and batch alignment. Scenario B should benefit from data integration, as the batches are orthogonal to the cell types and traditional baseline tools (e.g.,PCA) perform poorly in this situation (See Table 2.5). On the other hand, Scenario C represents the opposite, adversarial situation where data integration is not encouraged, as cells and batches are highly aligned, i.e.,the cells of different types are almost separated by batches already.

2.3.3 Numerical Experiments and Code

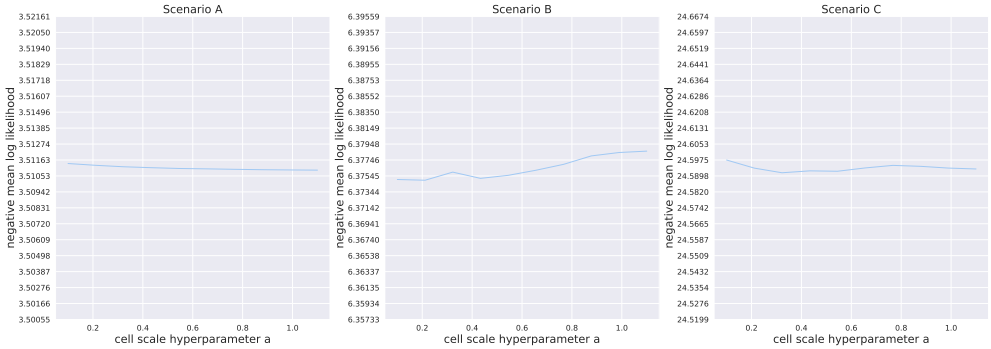
To study data integration, we merge the corresponding data sets (keeping the intersection of genes across data sets) according to Scenarios A-C. Pre-processing of the single-cell data sets is done using Scanpy [59] and Scanorama [39], with details described in Section 2.6.1. Our computational implementation of IHPF builds upon the original Python code for HPF in Ref. [48], and we use default hyper-parameters set in that reference. The code implementing IHPF is available at `https://github.com/barahona-research-group/scIHPF`.

Robustness of IHPF to hyper-parameters We have checked the robustness of IHPF under changes of two (of its six) hyper-parameters: the cell shape parameter a and the gene shape parameter c. The other hyper-parameters (a', b', c', d') are fixed to the values recommended in the scHPF package [48], and the IHPF noise ratio is set to $\alpha = 0.1$ for this robustness check. We run IHPF for the three data integration scenarios varying the hyper-parameters a and c between $[0.1, 1.1]$. In all cases, we extract $K = C$ factors, where C is the number of cell types present in the data. To evaluate the performance of model, we compute the negative mean log-likelihood of counts, under the Poisson assumption. Throughout this chapter, we will always be dealing with Poisson log-likelihoods. Figure 2.2 shows that the performance of IHPF in all the data integration scenarios is highly robust to changes in the hyper-parameters a and c. Once the robustness of IHPF to the hyper-parameters is established, we set $a = c = 0.3$ as suggested in the scHPF package [48] and keep these parameters fixed throughout the rest of the chapter.

2.4 Results

In this section, we first consider the effect of the noise ratio for data scenarios with different alignment between cell types and batches. We then compare the quality of the latent factors obtained with different methods for the purpose of clustering cell types and visualisation. Finally, we show that IHPF can extract

(a) Robustness under change of cell scale hyper-parameter, a



(b) Robustness under change of gene scale hyper-parameter, c

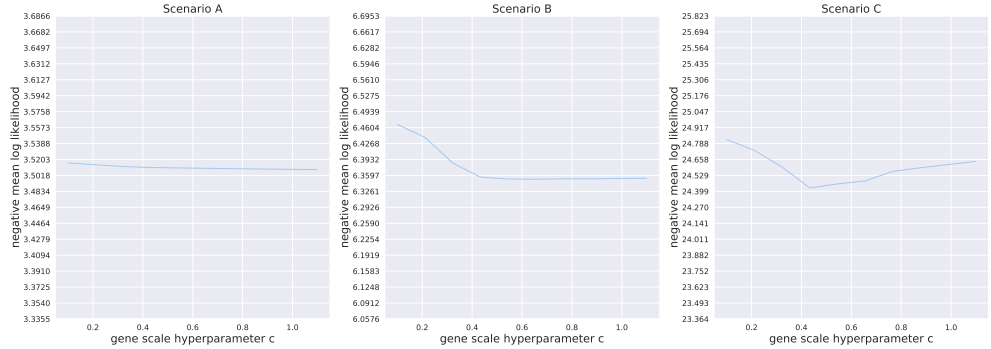


Figure 2.2: Negative Mean log-likelihood of IHPF for the three scenarios with varying hyper-parameters: (a) cell scale hyper-parameter, a; (b) gene scale hyper-parameter, c. For all scenarios, varying the hyper-parameters has almost no effect on the model performance.

sparse latent factors with a dual block structure with respect to both gene and cell spaces, thus enhancing their interpretability.

2.4.1 Data integration of scRNA-seq data sets using IHPF

The noise ratio α in IHPF (see Algorithm 8) encodes how much of the observed data variance is attributed to the shared biological signal (i.e., cell types) or to batch-specific differences. When the noise ratio becomes zero, the batch labels are ignored and IHPF recovers the corresponding non-integrative result (i.e., HPF). Therefore, the noise ratio needs to be tuned for the specific data integration scenarios: a noise ratio that is too large will result in over-correction, so that the cell scores will not capture the biological differences between cell types, whereas a noise ratio that is too small will result in extra cell clusters whereby cells with similar biology could be separated into smaller groups by batch.

Prior knowledge could be used to guide the selection of the noise ratio, e.g., experimental design or specific biological hypotheses for a particular analysis scenario can be indicative of the size and relevance of batch effects. Here, we do not assume any such information and we explore computationally how the noise ratio can be used to aid with the extraction of more meaningful biological representations of cell types in the presence of different types of alignment between biological and batch labels in the data. We run a grid search on the noise ratio, scanning 8 different values of the noise ratio

$\alpha = (0.0001, 0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 1.0)$ and we compute the quality of clusterings obtained from IHPF latent factors against both cell and batch labels, as well as in terms of an intrinsic measure. Specifically, we calculate the Adjusted Mutual Information (AMI) of clusters against cell types and batch labels, and the Silhouette coefficient (SC), an intrinsic measure of the quality of a clustering without knowledge of ground truth labels. For a more extensive discussion and explanation of these measures and the clustering problem, see full results in Section 2.4.2 below.

Table 2.3: Quality of clusters obtained from IHPF with different noise ratios. Adjusted Mutual Information (AMI) between the clustering obtained from latent factors *versus* cell types and batches are calculated. Silhouette coefficient (SC) of the clusterings obtained by applying k-means clustering to K normalised cell latent factors obtained from five matrix factorisation models. Here, $k = K = C$. Results are reported over 5 different random seeds.

Scenario	Noise ratio (α)	Cell AMI	Batch AMI	SC
A	0.0001	**0.981** ± 0.001	0.498 ± 0.000	0.942 ± 0.006
	0.0010	0.979 ± 0.001	0.498 ± 0.000	**0.963** ± 0.004
	0.0100	0.978 ± 0.001	0.498 ± 0.000	0.951 ± 0.001
	0.1000	0.914 ± 0.004	0.450 ± 0.003	0.818 ± 0.003
	0.2500	0.508 ± 0.012	0.179 ± 0.007	0.647 ± 0.004
	0.5000	0.206 ± 0.001	0.026 ± 0.000	0.587 ± 0.001
	0.7500	0.160 ± 0.005	0.012 ± 0.001	0.580 ± 0.000
	1.0000	0.122 ± 0.048	0.009 ± 0.003	0.582 ± 0.014
B	0.0001	0.620 ± 0.044	0.151 ± 0.039	0.528 ± 0.047
	0.0010	0.628 ± 0.025	0.180 ± 0.037	0.525 ± 0.043
	0.0100	**0.630** ± 0.028	0.151 ± 0.032	0.502 ± 0.039
	0.1000	0.597 ± 0.042	0.121 ± 0.021	0.479 ± 0.033
	0.2500	0.620 ± 0.024	0.109 ± 0.022	**0.532** ± 0.029
	0.5000	0.622 ± 0.033	0.079 ± 0.036	0.510 ± 0.016
	0.7500	0.575 ± 0.059	0.145 ± 0.071	0.488 ± 0.048
	1.0000	0.587 ± 0.051	0.099 ± 0.015	0.524 ± 0.050
C	0.0001	**0.643** ± 0.027	0.664 ± 0.035	**0.521** ± 0.024
	0.0010	0.625 ± 0.048	0.621 ± 0.054	0.474 ± 0.031
	0.0100	0.572 ± 0.054	0.552 ± 0.031	0.426 ± 0.039
	0.1000	0.433 ± 0.057	0.406 ± 0.048	0.338 ± 0.028
	0.2500	0.214 ± 0.056	0.211 ± 0.050	0.248 ± 0.035
	0.5000	0.141 ± 0.038	0.146 ± 0.035	0.232 ± 0.009
	0.7500	0.086 ± 0.048	0.100 ± 0.050	0.217 ± 0.023
	1.0000	0.087 ± 0.024	0.103 ± 0.029	0.233 ± 0.011

Table 2.3 presents these three metrics for the clusterings obtained from the IHPF latent factors computed with varying values of α. The results confirm the expected characteristics of each data Scenario. The 'easy' Scenario A performs well in assigning cell types (high cell AMI) and benefits from a modest amount of data integration. The beneficial effects of data integration are best seen in Scenario B, where the quality of the clusterings is high under values of α up to 0.5, whereas Scenario C performs best with minimal data integration, i.e., with low values of α. Therefore, when data integration is beneficial (Scenario B), setting the noise ratio to a larger value can reduce batch effects (batch AMI) without a tradeoff in learning the biological difference between cell types (cell AMI). These results are consistent with our biological knowledge of the datasets: batches in Scenario A are similar to each other; batches in Scenario C are highly aligned with the cell types; and batches in Scenario B are both technically distinct and containing cell mixtures. Therefore data integration is more important in this last case.

It is desirable to establish a protocol to select the noise ratio appropriately in the absence of biological

knowledge and without relying on metrics that use external labels (e.g., AMI). Table 2.3 shows that choosing noise ratios that lead to high values of the SC, an intrinsic measure of a well-separated clustering, broadly correspond to high AMI cell scores, thus supporting the selection of a noise ratio that leads to clusterings with high SC, In situations where a fast evaluation is desirable, we recommend to set the noise ratio of IHPF to 0.0001 (or any other small non-zero value). This choice implies weaker *a priori* assumptions about the compatibility of the datasets that are being integrated but, crucially, still allows data integration through the learning process in the algorithm. Indeed, the noise ratio can be interpreted as an initial guess of the degree of batch variability (noise) in the data, but the level of integration is learnt by the algorithm through the optimal batch specific scores. When only small batch corrections are required, the batch specific effects scores stay small throughout the learning. On the other hand, for large batch effects, the batch specific scores gradually grow to capture this variability as we run the algorithm. Therefore, the noise ratio acts as a parameter to guide the learning of batch scores, but does not preclude their eventual size. As a consequence of the presence of these additional parameters to be learnt, the cell scores obtained from IHPF and HPF are different. For example, for Scenario B, the average cross correlation between the cell scores of HPF and IHPF ($\alpha = 0.01$) is 0.606, ranging from 0.228 to 0.848. Even for the small values of α in Scenario C, the average cross correlation between the cell scores of HPF and IHPF ($\alpha = 0.0001$) is 0.714, ranging from 0.542 to 0.985.

2.4.2 Using IHPF latent factors for cell clustering and visualisation

Dimensionality reduction is a key step in computational pipelines for (unsupervised) clustering of scRNA-seq data sets. Ideally, one aims to extract a few latent factors that recapitulate the cell labels as fully as possible, and use the latent factors as data-driven coordinates on which to base classification, clustering, visualisations or coarse-grained descriptions that relate cell phenotypes to coherent groups of genes. PCA or NMF factors have had only limited success with scRNA-seq data due to the high levels of noise and zeroes in the data [60, 40].

To circumvent such limitations, complex pipelines have been developed including the use of similarity graphs and graph-based partitioning [61, 39, 62, 63]. However, such methods encounter problems under data integration scenarios since the latent factors could capture mostly batch information, and in that case the clusters would fail to recapitulate the cell types. Rather than considering only complex multi-step computational pipelines with many parameters, we examine here if relatively simple data integration methods can extract latent factors that capture cell type variability in the presence of batch effects.

We evaluate five widely-used methods for dimensionality reduction and data integration based on different principles: Scanorama, BBKNN, scVI and INMF, along with our proposed method IHPF. We also run two additional tools for scRNA-seq that does not perform data integration, namely HPF and PCA as additional baseline methods to compare. The code for scVI [38], Scanorama [39] and BBKNN [49] were all downloaded from their corresponding Github pages, and we use recommended parameters and apply pre-processing steps on the count matrices following the recommendations by the authors. For INMF [50] we have coded a Python implementation compatible with scikit-learn, and set the noise ratio to the default value of 1.0, as suggested by the authors. For INMF and IHPF, we do not apply any pre-processing, other than the necessary quality control measures. For PCA, we use the Python implementation in scikit-learn.

Cell clustering from cell latent factors We use each of the dimensionality reduction methods to extract $K = C$ factors, where C is the number of cell types present in the data, and the row-normalised

cell scores (e.g, from matrices W or Θ) are used to cluster the cells using k-means with $k = C$.

We first evaluate if the extracted cell latent factor scores induce well-separated clusters using the silhouette coefficient (SC), the (bounded) ratio of intra- and inter-cluster distances ($-1 \leq \text{SC} \leq 1$). For all methods other than BBKNN, SC is computed based on the cell latent factors (with latent dimension equal to the number of cell types in the scenario). For BBKNN, SC is computed based on the distance graph generated by the algorithm. For IHPF, we present results with noise ratios selected by two different methods: (i) selecting the optimal value by SC, denoted α^*_{SC}; and (ii) the default value $\alpha = 0.0001$.

Table 2.4 shows that, for all scenarios, IHPF gives the best or second-best separated clusters, as measured by the SC. On the other hand, methods that rely on Gaussian latent spaces and PCA, such as scVI and BBKNN give the least well-separated clusters, due to the mapping to non-sparse continuous variables. Scanorama and INMF provide an intermediate case, as a result of the graph-induced sparsity and the part-based property, respectively. The SC is, however, an intrinsic measure of cluster separability, but does not evaluate the quality of the clusters with respect to the underlying cell types or, alternatively, to the batch labels.

Table 2.4: Silhouette coefficient (SC) of the clusterings obtained by applying k-means clustering to K normalised cell latent factors obtained from five matrix factorisation models. Here, $k = K = C$. Results are reported over 5 different random seeds.

Method	Scenario A (ID 1-3)	Scenario B (ID 4-8)	Scenario C (ID 9-16)
PCA	0.860	0.345	0.410
Scanorama	0.455	0.185	0.220
BBKNN	0.004	0.021	-0.056
scVI	0.169 ± 0.011	0.171 ± 0.007	0.068 ± 0.007
INMF	0.458 ± 0.000	-0.249 ± 0.000	0.446 ± 0.001
HPF	0.931 ± 0.000	0.436 ± 0.072	0.549 ± 0.030
IHPF ($\alpha = 0.0001$)	0.942 ± 0.006	0.528 ± 0.047	0.521 ± 0.024
IHPF ($\alpha = \alpha^*_{SC}$)	0.963 ± 0.004	0.532 ± 0.029	0.521 ± 0.024

To quantify the quality of the clusterings with respect to the labels, we compute the Adjusted Mutual Information (AMI) between the cell clustering (i.e., k-means clusterings obtained from the cell scores) with respect to batch labels and cell types (Table 2.5). The AMI of the cell clusterings with respect to the batch labels (Table 2.5A) varies substantially for the different clustering methods and scenarios, but in general we observe that only IHPF is able to reduce batch effects in an adaptive manner, i.e., it only eliminates genuine batch effects that are not aligned with the cell types. This desirable effect can be seen by comparing the AMI of IHPF clustering *vs.* batches to the AMI of true cell types *vs.* batches, which is best captured by IHPF across all three scenarios. The other methods either over-correct or under-correct batch effects, and fail to adapt to the underlying variation in the alignment of batches and cell types.

Our main objective is to obtain a clustering from the latent factors that compares well with the cell labels (i.e., it extracts the biological signal in the presence of batch effects). The AMI of the cell clusterings *vs.* cell types (Table 2.5B) shows that IHPF has the overall best performance across the three scenarios, and can control the level of integration even in adversarial situations such as Scenario C, where data integration is not encouraged. In fact, IHPF is the only method that provides better AMI of cell clusterings *vs.* cell types than the baseline method PCA. Notably, IHPF achieves these results with a simple model specification with a small number of parameters (especially compared to complex pipelines or deep learning methods such as Scanorama or scVI).

Table 2.5: Adjusted Mutual Information (AMI) between the clustering obtained from latent factors *versus (a)* batches and *(b)* cell types. All the methods are applied without any gene selection. k-means is applied to K normalised cell latent factors obtained from each method, where the number of clusters k is set equal to the number of cell types C, i.e., $K = k = C$ (see Table 2.2). When the algorithms are dependent on random initialisations, the results are reported over 5 different random seeds (average and standard deviation)

(a) AMI of cell clustering *vs.* batch labels

Method	Scenario A (ID 1-3)	Scenario B (ID 4-8)	Scenario C (ID 9-16)
PCA	0.498	0.565	0.605
Scanorama	0.496	0.314	0.615
BBKNN	0.130	0.291	0.407
scVI	0.496 ± 0.004	0.098 ± 0.013	0.205 ± 0.026
INMF	0.170 ± 0.000	0.259 ± 0.000	0.297 ± 0.000
HPF	0.498 ± 0.000	0.245 ± 0.029	0.644 ± 0.021
IHPF ($\alpha = 0.0001$)	0.498 ± 0.000	0.151 ± 0.039	0.664 ± 0.035
IHPF ($\alpha = \alpha^*_{SC}$)	0.498 ± 0.000	0.109 ± 0.022	0.664 ± 0.035
AMI cell types vs. batches	*0.498*	*0.032*	*0.842*

(b) AMI of cell clustering *vs.* cell types

Method	Scenario A (ID 1-3)	Scenario B (ID 4-8)	Scenario C (ID 9-16)
PCA	0.972	0.445	0.616
Scanorama	0.969	0.433	0.556
BBKNN	0.158	0.383	0.394
scVI	0.970 ± 0.006	0.581 ± 0.005	0.217 ± 0.003
INMF	0.407 ± 0.000	0.276 ± 0.000	0.323 ± 0.000
HPF	0.981 ± 0.000	0.577 ± 0.033	0.617 ± 0.023
IHPF ($\alpha = 0.0001$)	0.981 ± 0.000	0.620 ± 0.044	0.643 ± 0.026
IHPF ($\alpha = \alpha^*_{SC}$)	0.979 ± 0.000	0.620 ± 0.024	0.643 ± 0.026

From our results, we also see that IHPF provides an improvement over both HPF (through data integration) and INMF (through the enhanced power of Poisson factorisation). Although the INMF results in Table 2.5 are obtained with the recommended value of 1 or the INMF noise ratio, the improved performance of IHPF over INMF is consistent over different values of this noise ratio (see full results in Table SM1 in the Supplementary Material). Furthermore, the improvement of IHPF over HPF is significant ($p < 0.05$) in Scenarios B and C, as seen in Table 2.6, where we report the p-values of a one-sided Brunner Munzel (BM) Test [64] to test the null hypothesis $\mathcal{H}_0$: *IHPF cell AMI is less than HPF cell AMI* against the alternative hypothesis $\mathcal{H}_1$: *IHPF cell AMI is greater than HPF cell AMI.*

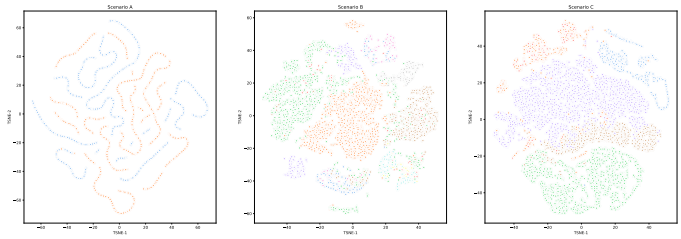
Table 2.6: p-values of one-sided Brunner Munzel Test comparing cell scores from IHPF and HPF. Results from control Scenario A is not reported as both IHPF and HPF have near identical performances.

	Scenario B	Scenario C
IHPF ($\alpha = 0.0001$)	0.0224	0.0464
IHPF ($\alpha = \alpha^*_{SC}$)	0.0152	0.0464

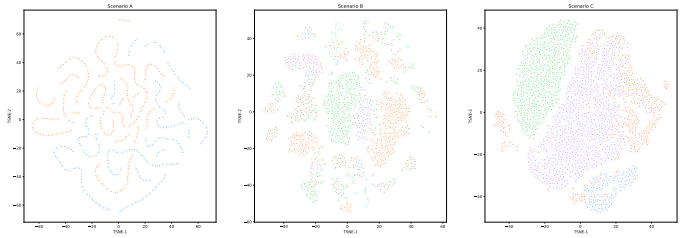
Visualisation based on cell latent factors Another important use of latent factors is as part of visualisation pipelines used for biological exploration of data sets. Typically, the latent factors extracted with a dimensionality reduction method are used as inputs to nonlinear mapping algorithms (such as TSNE or UMAP) to obtain representations of the data on the plane. Such planar representations are broadly used to display high dimensional data and to guide biological hypotheses. The quality of the latent factors is thus an important factor to obtain useful planar representations.

As a brief illustration of the benefits of data integration, we compare the visualisations obtained with TSNE from the different factors extracted through the dimensionality reduction methods, starting from PCA, as a simple baseline without batch correction, compared to TSNE mappings using IHPF factors and other methods. (Figure 2.3) Our results show that IHPF-based TNSE projections have better separated cell groups, which are less affected by potentially conflicting grouping by batches, especially in the case of Scenario B. This can be observed in the same projections coloured by batch label (Fig. 2.8 in the Supplementary Information), where we observe how the IHPF-based TSNE projections induce spatial groupings where cells of the same type but of different batches lie in close proximity, in contrast with the PCA groupings that are highly concomitant with batch labels. To see the same visualisations coloured according to batch type, see Section 2.6.3 in the Supplementary Information.

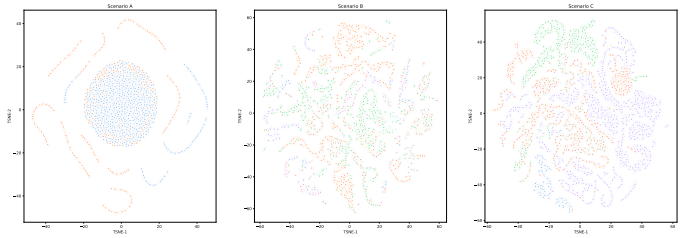
(a) TSNE from IHPF factors (coloured by cell type)



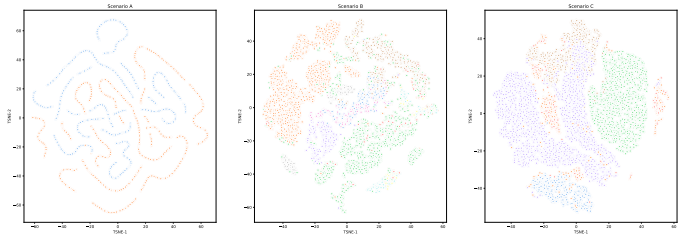
(b) TSNE from PCA components (coloured by cell type)



(c) TSNE from INMF components (coloured by cell type)



(d) TSNE from HPF components (coloured by cell type)



(e) TSNE from scVI components (coloured by cell type)

(f) TSNE from Scanorama components (coloured by cell type)

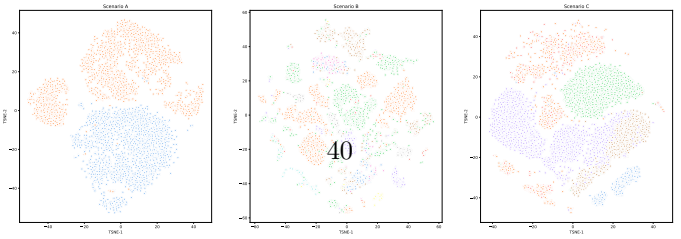


Figure 2.3: TSNE projections of Scenarios A-C obtained from latent factors from (a) IHPF, (b) PCA, (c) INMF, (d) HPF, (e) scVI and (f) Scanorama and coloured by cell type. TSNE plots with default

2.4.3 IHPF induces block-structured factors in both gene and cell spaces

In biological applications, it is often desirable that the latent factors are both descriptive and interpretable. Ideally, a few, sparse factors should be able to recapitulate the original data. In this context, a sparse factor is one that has high scores concentrated in a few variables and low (or zero) scores for the remaining variables; hence the score matrices would have a well defined *block structure*. Such sparse factors can be the basis for clusters with improved biological interpretability, because they can link cell types to a reduced set of characteristic genes.

Most dimensionality reduction methods (e.g., PCA) are not designed to induce sparsity in the latent factors. On the other hand, IHPF is characterised by an intrinsic sparsity that makes its factors good candidates for improved interpretability. To explore this idea in more detail, we examine the block structure in the matrices of both cell and gene latent factor scores obtained with different methods.

Block-structure in cell space Figure 2.4 shows the *cell scores* for Scenarios A-C obtained with PCA, INMF and IHPF. The block-structure present in all these matrices is a reflection of the fact that the cell latent factors capture information about cell types; indeed, these cell score matrices were the starting point for the clustering and visualisation results presented in Section 2.4.2.

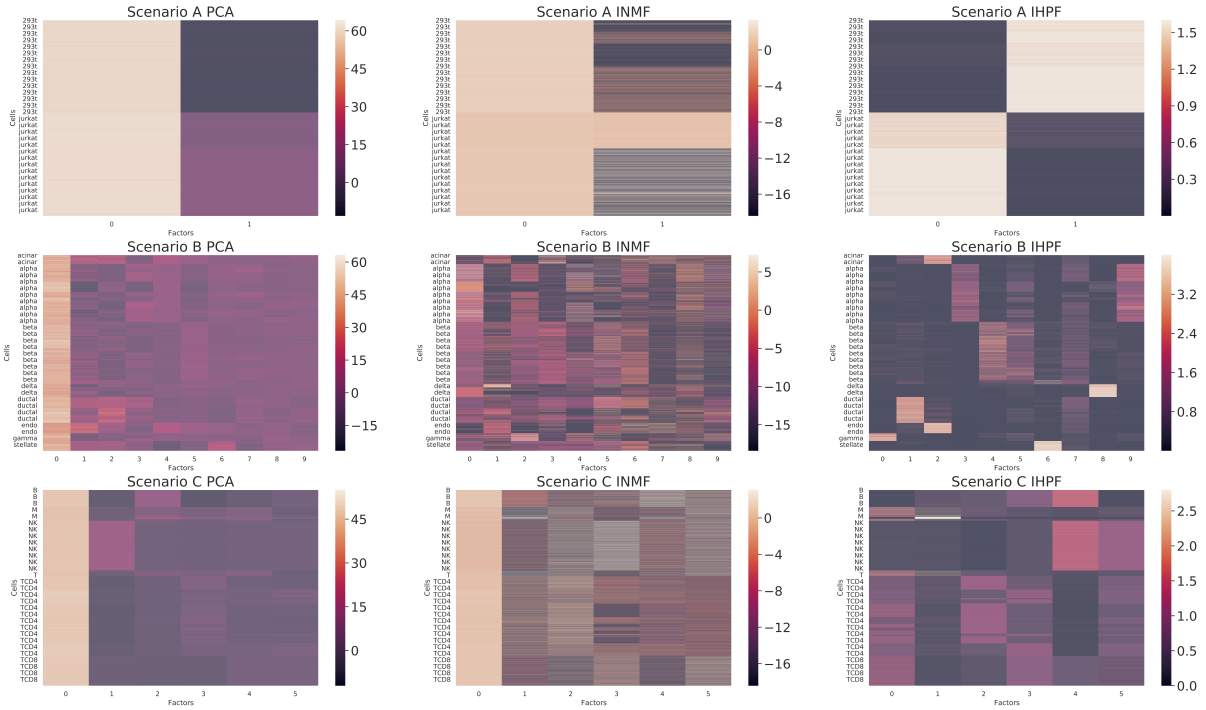


Figure 2.4: Cell latent factor scores ($N \times K$ matrices) obtained from PCA, INMF and IHPF for Scenarios A-C. The rows of the matrices correspond to cells sorted by cell type and the columns correspond to the latent factors. For each scenario, the number of latent factors is equal to the number of cell types ($K = C$). The block structure of the matrices is well aligned with the cell types, especially for IHPF.

Indeed, the silhouette coefficients (SC) for the k-means clusterings obtained from the cell scores of these three methods (Figure 2.5) confirm that IHPF cell scores lead to well separated clusters in all Scenarios, and PCA and INMF can also produce reasonable separability (especially for Scenario C, where data integration is not crucial).

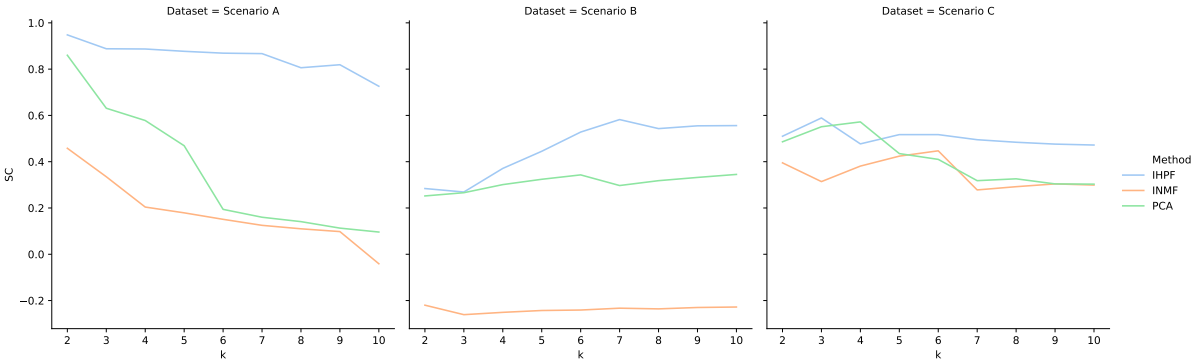


Figure 2.5: Silhouette coefficient (SC) of k-means clusterings (as a function of k) obtained from the $N \times K$ cell scores of PCA, INMF and IHPF for Scenarios A-C. Well separated clusterings, indicated by high SC values, are obtained from IHPF cell scores in all Scenarios.

Block-structure in gene space The results above examine the structure of the latent factor scores in cell space, but in many cases it is advantageous to induce clusterings that also display high concentration and explainability in gene space. We have thus analysed the k-means clusterings obtained from the $K \times M$ matrices of gene scores (e.g., the matrix of common gene scores B in the case of IHPF). In contrast with the situation observed for the cell scores in Figure 2.4, we find that only the IHPF *gene scores* exhibit a well defined block structure across all three data scenarios (Figure 2.6). Consequently, the SC in Figure 2.7 shows that only IHPF gene latent factors induce well-separated gene clusters associated with the extracted latent factors. In summary, IHPF learns latent factors that have a dual block-structure, in both cell and gene spaces with the potential for enhanced explainability and biological interpretability by linking cell types to gene clusters, as explored below.

Biological homogeneity of extracted gene clusters To characterise further the biological interpretability of the gene clusters extracted with IHPF, we proceed as follows. We start by selecting the number of clusters that gives the highest Silhouette coefficient (SC) of the gene scores (Figure 2.7). For Scenario A, the optimal number of gene clusters is 3. For Scenario B, the optimal number of gene clusters is 8. For Scenario C, the optimal number of gene clusters is 10. Note that the number of gene clusters is different from the number of cell types, as there is no one-to-one mapping between cell types and gene groups—a cell type can be linked to multiple gene groups and *vice versa*.

We then compute the biological homogeneity index (BHI) [66, 67] of the obtained gene clusterings. The BHI is a measure of within-cluster biological similarity of the Gene Ontology (GO) terms associated with the genes present in each cluster. Specifically, for a clustering $\{C_k\}_{k=1}^K$ with K clusters C_k, the BHI is given by:

$$\mathrm{BHI}\left(\{C_k\}_{k=1}^K\right) = \frac{1}{K}\sum_{k=1}^{K} \frac{1}{n_k(n_k-1)} \sum_{\substack{i,j\in C_k \\ i\neq j}} S(i,j)$$

where n_k is the size of cluster C_k, and $S(i,j)$ is the biological similarity between gene i and gene j, obtained using an information-theoretic semantic similarity based on GO terms. The BHI gives a score between 0 and 1 (see Ref.[68] and Section 2.6.2 in the Supplementary Information). We then compute the BHI of 10 randomised surrogate clusterings with the same number and sizes of clusters. We then compute the z-score of the BHI of the IHPF clustering against the ensemble of random clusterings to obtain a level of significance for the obtained clustering [69]. The results in Table 2.7 show that the clusterings obtained

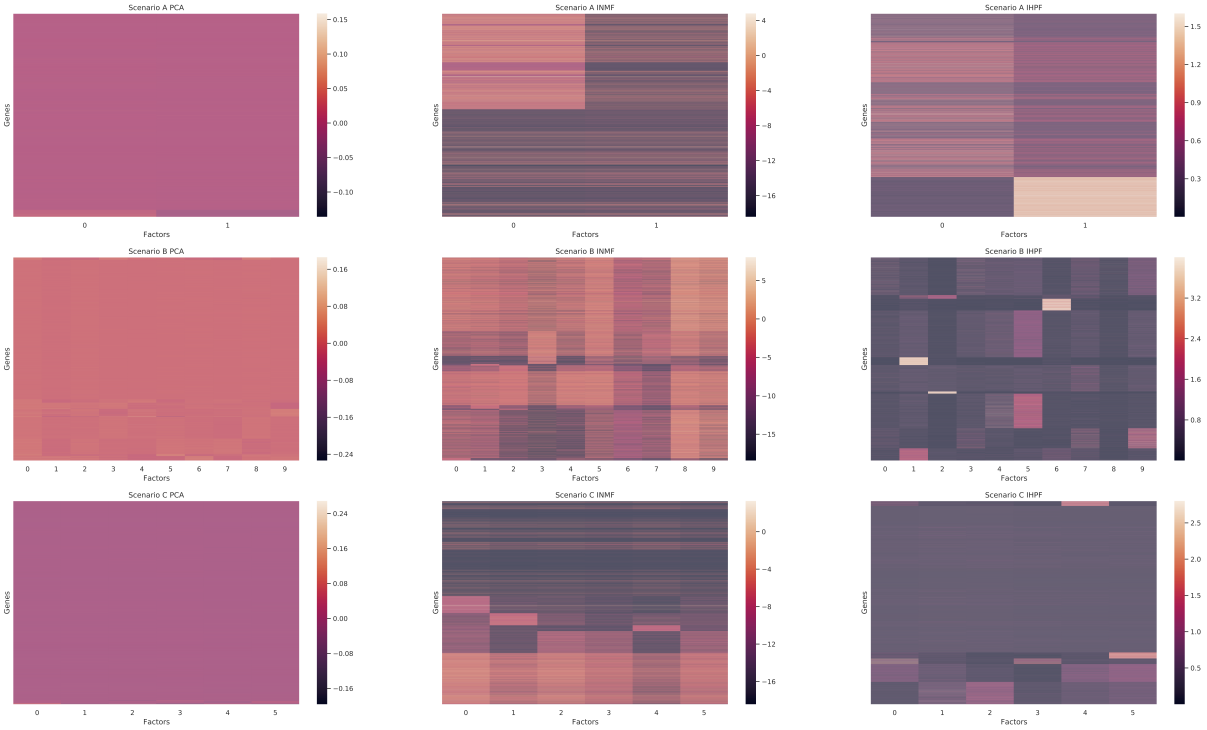


Figure 2.6: Gene latent factor scores ($K \times M$ matrices) obtained from PCA, INMF and IHPF for Scenarios A-C. As above, $K = C$. The rows (genes) are ordered using spectral co-clustering for ease of visualisation. There is only well-aligned block structure for the IHPF gene scores.

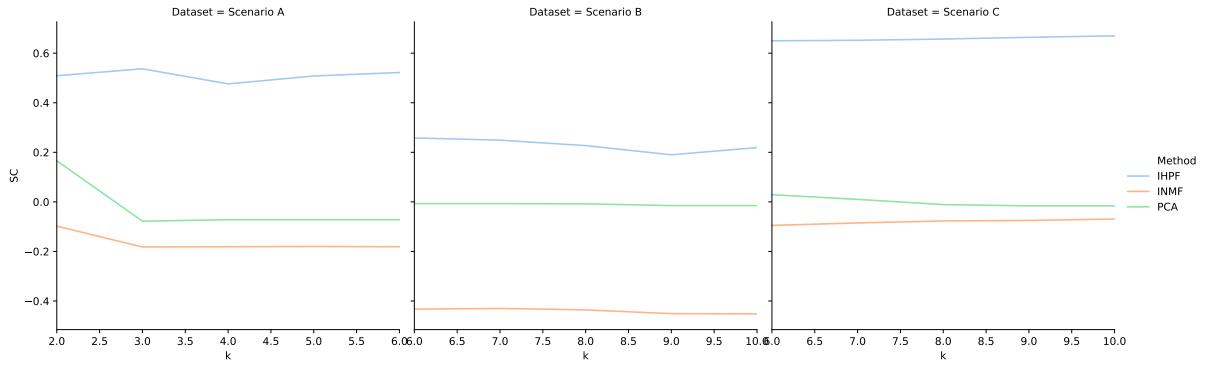


Figure 2.7: Silhouette coefficient (SC) of k-means clusterings (as a function of k) obtained from the gene scores of PCA, INMF and IHPF for Scenarios A-C. Only IHPF gene factors lead to well separated clusters with high SC values whereas INMF and PCA gene factors lead to poor clusters with negative values of SC.

from the IHPF gene scores for all Scenarios significant with respect to the randomised clusterings, with all z-scores greater than 2. Hence the gene clusterings correspond to meaningful biological groupings of genes. Furthermore, we also find that the BHI of gene clusterings obtained with IHPF is significantly higher than those obtained with HPF, as given by obtained a one-sided Brunner Munzel Test [64], with significant p-values below 0.05 for all scenarios when testing the null hypothesis that the BHI from IHPF is less than or equal to those from HPF. The significant improvement in BHI indicates that the gene scores learnt from IHPF are not only different to those learnt from HPF, despite the small noise ratio being set initially, but also have higher biological consistency. These results highlight the qualitative

difference introduced in the learning process of IHPF, as compared to HPF, since batch specific gene factors can be assigned to absorb noise in the datasets, thus allowing for an increase in the biological content of the gene factors.

Table 2.7: The BHI score of gene clusterings obtained from IHPF comapared to those obtained from HPF and a randomised baseline. The gene clusterings were obtained using k-means with $k = 3, 8, 10$ for Scenarios A, B and C, respectively. The noise ratio of IHPF is set to 0.0001. Results are reported over 20 random seeds.

	Scenario A	Scenario B	Scenario C
BHI of Randomised ensemble	0.064 ± 0.001	0.071 ± 0.001	0.065 ± 0.001
BHI of HPF	0.0696 ± 0.00003	0.0833 ± 0.0022	0.0773 ± 0.0011
BHI of IHPF ($\alpha = 0.0001$)	0.0704 ± 0.0002	0.0849 ± 0.0035	0.0805 ± 0.0015
z-score (IHPF > Randomised)	12.30	14.69	15.45
p-value (IHPF > HPF)	$\simeq 0$	0.0496	3.331 e-12

2.5 Discussion

Here we have presented a computational method to carry out dimensionality reduction and data integration for single-cell RNAseq data based on integrative hierarchical Poisson factorisation (IHPF). The proposed algorithm extends HPF and performs data integration by modelling the technical noise between data sets with the use of latent factors adapted to batch labels.

As in other data integration methods (e.g., INMF), the noise ratio hyper-parameter can be tuned to accommodate the amount of variability ascribed to batch effects *vs.* biological effects (i.e., cell types). When the noise ratio becomes negligible, the batch labels are ignored and IHPF recovers the results of HPF. The noise ratio can be adjusted depending on the different experimental data integration scenarios. In cases when there is good overlap of cell types across batches and a large set of common genes (Scenario A), or the other extreme when batches and cell types are highly aligned with little overlap of cell types across batches (Scenario C), data integration brings little advantage. In that case, IHPF indicates that a close to zero noise ratio is appropriate, and still provides very good performance. On the other hand, in more complex situations (where batches are technically different and contain a varied mixture of cells), IHPF offers a flexible and controlled way to correct the effect of batch effects and can outperform other methods. The robustness of IHPF under different data integration scenarios is useful when there is lack of biological prior knowledge of the datasets. IHPF exhibits good performance across extremes of alignment of cell types and batches, being able to control intrinsically the level of data integration from scenarios where data integration is not required to well mixed datasets that require strong batch correction. We also provide a default value (0.0001) for IHPF which gives good performances, which is useful when computational resources are constrained so that grid search on noise ratios cannot be performed. We shown even with a very small default value, IHPF found better cell and gene scores than HPF across different scenarios that capture more biological information of the dataset.

We note that IHPF has no requirements on library sizes of the cells in the data sets, unlike PCA-based methods such as Scanorama, which requires pre-processing of cells to have normalised sizes. In addition, IHPF is shown to recover factors with dual block structure in both the cell and gene spaces, thus enhancing the explainability of latent factors in terms of groups of genes with biological content. We have made IHPF available as a Python package for use by the community.

There are several directions for extensions and improvement of this work. Currently, IHPF can only correct differences between data sets that are categorical in nature, such as batch effects. When the differences are continuous factors, such as population effects, a different model is needed. Further, the latent factors obtained with IHPF could also serve as the seed for more sophisticated classification and clustering methods, including variational autoencoders or graph-based multi-scale clustering methods [70]. In particular, in this work we have focused on problems where the number of cell types is given, and have not addressed the further question of unsupervised selection of the number of cell types in the data. This is an open area of research, of high topical importance in current efforts towards the construction of data-driven Cell Atlases. Our work indicates that IHPF factors could be used as the coordinates to compute distances between cells in lieu of, e.g., PCA factors, as seen by the good separation in the clusters and visualisations as well as the consistent findings when scanning k in the k-means algorithm. The cell-to-cell distances computed from IHPF factors could then be the input to unsupervised clustering algorithms where the number of clusters can be optimised according to, e.g., the robustness of similarity subgraphs [71, 63].

2.6 Supplementary Information

2.6.1 Data pre-processing for the different scRNA-seq methods

Our pre-processing starts with quality control on the data sets, whereby we filter cells with a low transcript count. For each of the computational methods, we follow the pre-processing steps recommended by the original authors using Scanpy [59]. Specifically, we use the following functions:

- Scanorama: `sc.pp.recipe_zheng17`
- scVI: `sc.pp.normalize_total`, `sc.pp.log1p`
- BBKNN: `sc.pp.recipe_zheng17`
- INMF: None
- IHPF: None
- HPF: None
- PCA: `sc.pp.normalize_total`, `sc.pp.log1p`

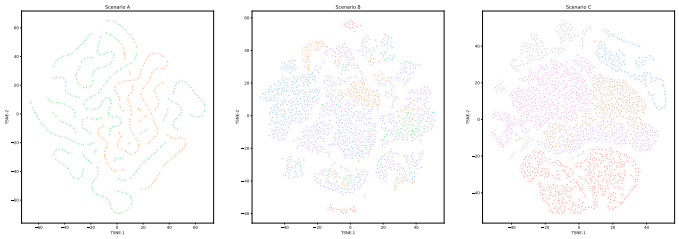
2.6.2 Computation of BHI

The genes in Scenarios A, B and C are mapped to GO terms using the org.Hs.eg.db annotation package. A total of 53.31% of genes in Scenario A, 95.26 % of genes in Scenario B and 53.31% of genes in Scenario C are mapped to a valid GO term. After that, the BHI is computed using our Python implementation of the function in the R package clValid [66]. In the calculation of biological similarity $S(i, j)$, GO terms with more than 1000 genes mapped are ignored as these GO terms are too generic to carry any meaningful biological information.

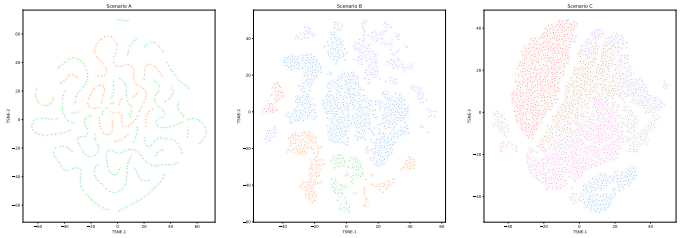
2.6.3 Additional Visualisations

Additional visualisations of data sets (with cells colored by batch labels) obtained from TSNE using different methods to generate the dimensionally reduced factors are shown in Figure 2.8.

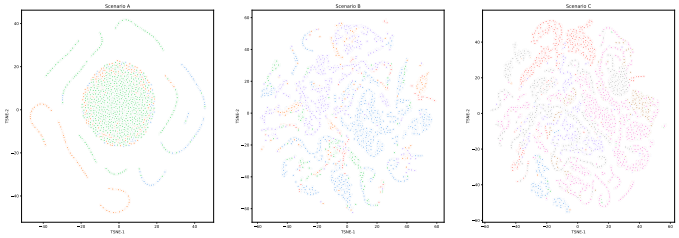
(a) TSNE from IHPF factors (coloured by batch labels)



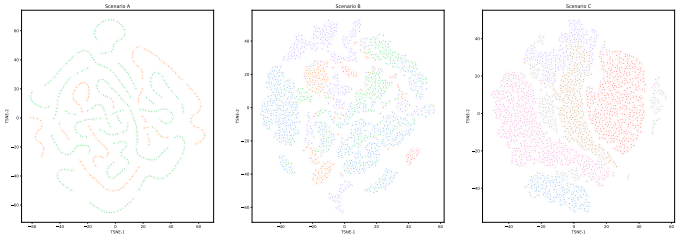
(b) TSNE from PCA components (coloured by batch labels)



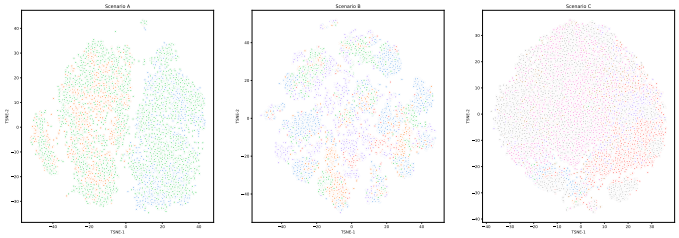
(c) TSNE from INMF components (coloured by batch labels)



(d) TSNE from HPF components (coloured by batch labels)



(e) TSNE from scVI components (coloured by batch labels)



(f) TSNE from Scanorama components (coloured by batch labels)

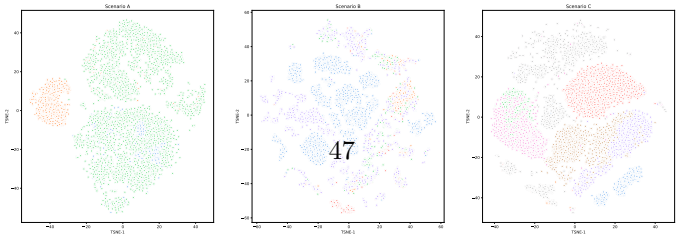


Figure 2.8: TSNE projections of Scenarios A-C obtained from latent factors from (a) IHPF, (b) PCA, (c) INMF, (d) HPF, (e) scVI and (f) Scanorama and coloured by batch label.

2.6.4 Clustering quality from INMF with different noise ratios

The noise ratio is set to 1.0 for INMF in the main text. We run INMF with 4 different noise ratios (0.001,0.01,0.1,1.0) and report the performance in Table 2.8. Over all wide range of noise ratios, none of the INMF models obtain better results than IHPF.

Table 2.8: Quality of clusters obtained from INMF with different noise ratios. Adjusted Mutual Information (AMI) between the clustering obtained from latent factors *versus* cell types and batches are calculated. Silhouette coefficient (SC) of the clusterings obtained by applying k-means clustering to K normalised cell latent factors obtained from five matrix factorisation models. Here, $k = K = C$. Results are reported over 5 different random seeds.

Scenario	Noise Ratio (α)	Cell AMI	Batch AMI	SC
A	0.001	0.352 ± 0.000	0.145 ± 0.000	0.465 ± 0.000
	0.010	0.358 ± 0.002	0.147 ± 0.001	0.462 ± 0.001
	0.100	0.371 ± 0.000	0.153 ± 0.000	0.457 ± 0.000
	1.000	0.407 ± 0.000	0.170 ± 0.000	0.458 ± 0.000
B	0.001	0.311 ± 0.000	0.282 ± 0.000	0.463 ± 0.001
	0.010	0.311 ± 0.001	0.283 ± 0.000	0.462 ± 0.000
	0.100	0.314 ± 0.000	0.285 ± 0.000	0.459 ± 0.000
	1.000	0.323 ± 0.000	0.297 ± 0.000	0.446 ± 0.001
C	0.001	0.407 ± 0.000	0.082 ± 0.000	-0.249 ± 0.000
	0.010	0.412 ± 0.000	0.084 ± 0.000	-0.233 ± 0.000
	0.100	0.392 ± 0.001	0.075 ± 0.000	-0.299 ± 0.000
	1.000	0.276 ± 0.000	0.259 ± 0.000	-0.228 ± 0.000

2.7 Background on computational tools for scRNA-seq analysis

In this section, we provide some background materials on the computational tools used in scRNA-seq analysis for the three different data integration scenarios in this chapter.

Existing challenges for tools used in single-cell analysis There are various challenges for the tools developed recently for single-cell data integration.

First, tools that are based on similar ideas can have different outcomes depending on the preprocessing steps applied and the exact programming implementation. It has been shown that the performance of PCA depends on data preprocessing steps taken [72]. Given the high dimensionality of the data, feature selection and other preprocessing steps are used to reduce the size of the gene space to be considered. Feature selection relies on choosing a suitable threshold to keep the highly variable genes (HVGs) in the data. It has been shown that the choice of threshold can affect downstream analysis and sometimes lead to different discoveries of differentially expressed genes (DEGs) [73].

Second, there is no consensus in choosing the probability distributions to model the count matrices of the number of reads mapped to each gene in each cell [74]. Some researchers suggest the need to distinguish different reasons leading to excess zeros, which are used to represent the missing values (dropouts), in scRNA-seq data [75]. Currently, the term "zero-inflation" in literature refers to both the biological effects of most genes not expressed at the single-cell level and the technical effects of not observing the counts due to missing data due to the nature of the sequencing process. Clarifying the origin of the sparsity in scRNA-seq data is important, as we would like to correct for sources of experimental errors, but not biological effects. Some models enforce sparsity on the data in a way that imposes strong biological assumptions, which do not always hold. For example, There are models that have a step that randomly creates zero observations, ignoring the actual underlying expression in cells [76]. For example, a model may implicitly introduce a systematic effect which drops all counted molecules from a particular gene in a particular cell. The use of zero-inflated distribution such as zero-inflated Poisson or zero-inflated Negative Binomial distribution is often unexplained with the hyper-parameters usually set to match the sparsity of the test datasets. It is suggested to use separate models for scRNA-seq analysis to separate the true gene expression (expression model) and the noise (measurement model) [75]. It is not clear from empirical evidence whether these more complicated distributions outperform simpler models, such as the Poisson distribution.

Data preprocessing Data preprocessing is an important step for data integration tools. Preprocessing can remove experimental effects from datasets, simplifying the requirements for subsequent analysis. Most preprocessing steps involve scaling and centring of data. It is also common to normalise the data by taking a non-linear transformation, such as taking a logarithm on the count data.

Newer research illustrates several drawbacks of preprocessing in data integration. Using a log transformation ($\log +1$) can distort the count data and introduce artificial variability to the data, especially in sparse datasets [72]. The choice of a particular data preprocessing routine reflects assumptions made on the underlying distribution of data. Many scRNA-seq analysis tools have preprocessing integrated into their pipeline, and therefore limit their use to datasets conforming to the assumptions embedded in the pipeline.

Measuring quality of clusters obtained from data integration tools The performance of data integration tools is measured by two major criteria: batch alignment, and cell type agreement, which

are both used in different studies, such as [70, 38]. A good data integration tool will balance these two criteria.

In scVI, the average negative entropy of batch composition of the k-nearest neighbours of each cell is used to measure batch alignment [70]. We then compute k-nearest purity, which is the average percentage overlap of k-nearest neighbours of each cell before and after integration to measure cell types agreement.

Alternatively, Adjusted mutual information (AMI) [77] can be calculated to compare batch labels with k-means clustering labels on the joint latent space [35]. A lower AMI score would indicate better mixing of datasets. Agreement on cell types can be measured by AMI similarly. AMI is calculated to compare cell labels with k-means or other appropriate clustering labels on the joint latent space. A higher AMI score indicates better separation between cells of different types.

Definition 1 (Adjusted Mutual Information)**.**
Given a set S of N elements with two partitions $U = \{U_1, \ldots, U_A\}$ of A clusters and $V = \{V_1, \ldots, V_B\}$ of B clusters. Entropy of a partition $H(U)$ and Mutual information between two partitions $MI(U, V)$ are defined as

$$H(U) = -\sum_{i=1}^{A} \frac{|U_i|}{N} \log \frac{|U_i|}{N}$$

$$MI(U, V) = -\sum_{i=1}^{A} \sum_{j=1}^{B} \frac{|(U_i \cap V_j)|}{N} \log \frac{N|(U_i \cap V_j)|}{|U_i||V_j|}$$

Adjusted Mutual Information $AMI(U, V)$ corrects for the chance of overlapping from two random clustering

$$AMI(U, V) = \frac{MI(U, V) - \mathbb{E}(MI(U, V))}{max(H(U), H(V)) - \mathbb{E}(MI(U, V))}$$

where the expectation is taken under the assumption of overlapping under a hyper-geometric model.

Benchmark tools for data integration Data integration and batch correction is a new challenge in the scRNA-seq analysis, as newer technologies and studies produce datasets that are not directly comparable with older ones. Differences in sequencing depth and instrument lead to various technical effects that need to be corrected. A review of existing tools for batch correction is given by [35] proposes five different scenarios for assessing data integration. Scenarios ranged from easier tasks such as integrating identical cell types from different technologies to merging multiple batches without a reference dataset. Performance is assessed by inspecting data visualisations by UMAP [78], clustering metrics on cell types, and batch labels. In the review, older methods that use empirical Bayes or (generalised) linear models for batch correction have been superseded by newer methods using graph and deep learning approaches. Most tools for integrating scRNA-seq datasets learn a joint latent space for the datasets (data integration) and correct the technical differences between datasets (batch correction). See Figure 2.9 for a typical workflow for various commonly used data integration tools. Table 2.9 lists a few commonly used tools with the dimension reduction and clustering algorithms used.

A good understanding of how to separate batch and biological effects in datasets allows for efficient data integration. Earlier tools assumed the differences between batches or datasets are undesirable technical effects, which need to be removed. They fail when heterogeneous datasets are combined. For example, when there is an unbalanced distribution of cell types, such that rare cell types are only found in

1. Data preprocessing: Scale and normalisation of count data, variable gene selection
2. Dimension reduction: Find joint latent factors in the cell and/or gene space
3. Clustering/Community detection: Joint clustering using cell latent factors/ Build anchors between datasets by Mutual Nearest Neighbours (MNN)
4. Batch correction: Correct differences between datasets to produce latent factors on the integrated dataset for downstream analysis
5. Data visualisation: Visualise cell space by manifold learning algorithms such as tSNE or UMAP

Figure 2.9: Common workflow of data integration and batch correction of scRNA-seq datasets

Method	Dimension reduction algorithm	Clustering algorithm	Year	Ref
INMF	Non-negative Matrix Factorisation	N.A.	2016	[50]
scVI	Variational Auto-encoder	k-nearest Neighbour	2018	[38]
BBKNN	PCA	Nearest Neighbour	2019	[49]
Scanorama	randomised SVD	Mutual Nearest Neighbour	2019	[39]

Table 2.9: Examples of data integration/batch correction tools.

some datasets or with a low count. In Scanorama, and other methods that use Mutual Nearest Neighbour (MNN) to merge various latent spaces, the following assumptions are often made [79].

- Each pair of batches shares at least 1 common cell type
- The batch effect is separable from the biological subspace
- That the batch effect variation is much smaller than the biological effect variation between different cell types

While recent tools can correct for batch effects, and provide a latent representation or corrected count matrix for downstream analysis, few can provide an interpretable breakdown of data into the parts that correspond to the batch, population, and biological effects.

Dimensionality reduction algorithms Dimension reduction algorithms studied in the thesis come from two different classes of algorithms, matrix factorisation and deep learning. Matrix factorisation refers to decomposing a given high dimensional matrix into two lower-rank matrices. Reconstruction is optimised with respect to a loss function for traditional matrix factorisation or likelihood for probabilistic matrix factorisation. For some special cases of loss functions, such as the Euclidean norm, a corresponding likelihood can be derived. Optimisation methods commonly used include gradient descent or multiplicative weight updates. For probabilistic-based factorisation, Variational Bayes (VB), Expectation Maximisation (EM), and Gibbs sampling are commonly used.

In table 2.10, we give different examples of matrix factorisation methods studied in this thesis. Matrix factorisation methods are also used in applications other than genomics. For user-recommendations data, numerous algorithms have been proposed which can incorporate user and item metadata [80]. Matrix factorisation methods can be classified by whether non-negative constraints are imposed on latent factors. In general, non-negative matrix factorisation methods give sparse representations and it is able to decompose the data into local structures.

Method	Modelling distribution	Inference Method	Implementation	References
PCA	Gaussian	Singular Value Decomposition	Python(Scikit-learn)	[44]
NMF	Poisson	Expectation–maximisation	Python(Scikit-learn)	[43]
scHPF	Poisson	Variational Bayes	Python	[48]

Table 2.10: Examples of matrix factorisation methods. Modelling distribution refers to the probabilistic distribution used to model counts. The inference method refers to the algorithm used to infer the parameters in different matrix factorisation models. Implementation refers to the programming language that is used to implement the algorithm.

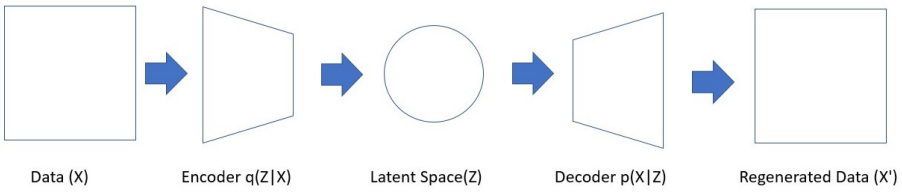


Figure 2.10: Schematic diagram for VAE.

Deep learning refers to the use of neural networks in model inference. Generative models in deep learning such as Variational Autoencoders (VAE) are used in scRNA-seq analysis. An example implementation of VAE for scRNA-seq datasets includes scVI [38] and others, with scVI performing better than other neural network models as shown by the authors [38].

VAE consists of two coupled but independently parameterised models, the recognition model (encoder) and the generative model (decoder). The encoder maps the input data to the lower dimensional latent space which is then used by the decoder to generate the reconstruction of data. See Figure 2.10 for the schematic representation of a VAE. The decoder learns the joint distribution $p_\theta(X, Z)$ which usually factorised into $p_\theta(Z)p_\theta(X|Z)$ with a prior distribution $p_\theta(Z)$ over latent space. The intractable posterior distribution $p_\theta(Z|X)$ is approximated by $q_\phi(Z|X)$, which is parameterised by a neural network [81].

2.7.1 Mathematical foundations of matrix factorisation methods

We provide details of the mathematical models and algorithms of some commonly used dimension reduction methods developed for scRNA-seq analysis. There are two families of methods. One of them is PCA-based methods (PCA, ZIFA, ZINB-WaVE), where the latent space is assumed to follow Gaussian distribution. The others are Non-negative matrix factorisation-based methods such as (NMF, HPF, pCMF, hGNB) where non-negative constraints are imposed on the latent space.

Consider a data set consisting of N samples, each of them described by $M \gg 1$ features. In the case of scRNA-seq, the N samples are cells of different types, and each of the M features is the transcription count of a particular gene. The data is compiled into a data matrix X of dimensions $N \times M$.

Matrix factorisation methods attempt to find a good approximation $\widetilde{X}$ of the data set in terms of two matrices W and V with reduced dimension (rank) $K \ll M$ such that their product is close to the original matrix:

$$X_{N\times M} \approx \widetilde{X} = W_{N\times K} V_{K\times M}. \tag{2.5}$$

PCA-based methods

It has been shown [44] that PCA can be understood from a probabilistic perspective as a generative model where the samples are noisy observations of a combination of latent factors that are multivariate Gaussian [45, 46]:

$$X_{N\times M} \sim \text{Gaussian}(WV, \sigma^2\mathbb{I}) \tag{2.6}$$

$$W_{N\times K} \sim \text{Gaussian}(\mathbf{0}, \mathbb{I}) \tag{2.7}$$

$$V_{K\times M} \sim \text{Gaussian}(\mathbf{0}, \mathbb{I}) \tag{2.8}$$

$$\sigma \sim \text{log-Normal}(0, 1). \tag{2.9}$$

Zero-inflated Factor Analysis (ZIFA) [82] is an extension of PCA which takes into account to over-abundance of zeros in scRNA-seq count datasets.

Given a non-negative matrix $X = [x_1, \ldots, x_N]$ of size $N \times M$, where N is the number of observations and M is the number of features. The data is assumed to be generated from a latent space matrix $Z = [z_1, \ldots, z_N]$ of size $N \times K$, where K is the number of latent factors.

The ZIFA model is shown in Figure 2.11 where $\mathcal{I}$ denotes $K \times K$ identity matrix, A is the $D \times K$ factor loading matrix, μ is the $D \times 1$ mean expression vector. $W = \text{diag}(\sigma_1^2, \ldots, \sigma_D^2)$ is the diagonal covariance matrix. p_0 represents the dropout probability which is modelled by $p_0 = \exp(-\lambda y_{ij}^2)$. λ is shared across different features to reduce the number of parameters to be estimated.

$$\begin{aligned}
z_i &\sim \text{Normal}(0, \mathcal{I}) \\
y_i|z_i &\sim \text{Normal}(Az_i + \mu, W) \\
h_{ij}|y_{ij} &\sim \text{Bernoulli}(p_0) \\
x_{ij} &= y_{ij} \text{ if } h_{ij} = 0 \\
&= 0 \text{ if } h_{ij} = 1
\end{aligned}$$

Figure 2.11: ZIFA model

Zero-Inflated Negative Binomial-based Wanted Variation Extraction (ZINB-WaVE) [83] uses a zero-inflated negative binomial distribution to model highly sparse scRNA-seq count datasets. It uses cell and gene-level covariates to model technical effects.

The zero-inflated negative binomial distribution is parameterised by the mean μ, dispersion θ and dropout probability π as follows. ZINB-WaVE models the count matrix Y_{ij} of $N \times M$ where N is the number of cells and M is the number of genes by a zero-inflated negative binomial distribution with the parameters given by generalised linear models based on cell and gene covariates. In figure 2.12 we show the model parameters. X is a cell co-variate matrix of $N \times C$, C is the number of cell covariates. V is a gene co-variate matrix of $M \times G$, G is the number of gene covariates. W is the unobserved cell-level co-variate of $N \times K$, K is the number of latent factors, O_μ, O_π are offsets matrices . η_j are the gene-specific dispersion parameters on log scale. $\beta_\mu, \beta_\pi, \gamma_\mu, \gamma_\pi, \alpha_\mu, \alpha_\pi$ are the regression coefficients.

Ridge regression is used to optimise the penalised likelihood and fit regression coefficients in the above model alternatively until the parameters converge. A key limitation is long computational time required for the convergence of model parameters, making this method not suitable for large datasets or data integration tasks.

$$f_{NB}(y|\mu,\theta) = \frac{\Gamma(y+\theta)}{\Gamma(y+1)\Gamma(\theta)}\left(\frac{\theta}{\theta+\mu}\right)^{\theta}\left(\frac{\mu}{\theta+\mu}\right)^{y}$$
$$f_{ZINB}(y|\mu,\theta,\pi) = \pi\delta_0(y) + (1-\pi)f_{NB}(y|\mu,\theta)$$

$$\log(\mu_{ij}) = \left(X\beta_\mu + (V\gamma_\mu)^T + W\alpha_\mu + O_\mu\right)_{ij}$$
$$\text{logit}(\pi_{ij}) = \left(X\beta_\pi + (V\gamma_\pi)^T + W\alpha_\pi + O_\pi\right)_{ij}$$
$$\log(\theta_{ij}) = \eta_j$$

Figure 2.12: ZINB-WaVE model

Non-negative matrix factorisation (NMF)

NMF aims to represent a high-dimension matrix as the product of two lower-rank matrices.

Given a non-negative matrix X of size $N \times M$, where N is the number of observations and M is the number of features. With a pre-determined number of latent factors K, we find two non-negative matrices (sparse) coding W of size $N \times K$, dictionary H of size $K \times M$ so that the difference between X and WH is minimised under a chosen loss function. The NMF optimisation problem can be solved by KKT conditions and multiplicative updates [84, 85, 86].

Proposition 2.7.1 (Karush-Kuhn-Tucker (KKT) conditions [85, 86])**.**
Karush-Kuhn-Tucker (KKT) conditions can be used to solve optimisation problems that involve inequality constraints. Given a function $f : \mathbb{R}^d \mapsto \mathbb{R}$ where the domain of a function is $\mathcal{D}$. The optimisation problem is defined as a minimisation problem subject to different equality and inequality constraints.

$$\min_x f(x)$$
$$y_i(x) \leq 0, 0 \leq i \leq m_1$$
$$h_i(x) = 0, 0 \leq i \leq m_2$$

To solve the optimisation problem, we first define the Lagrangian and the dual function.

Definition 2 (Lagrangian and dual variables)**.**
The Lagrangian function for the optimisation problem above is $\mathcal{L} : \mathbb{R}^d \times \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \mapsto \mathbb{R}$

$$\mathcal{L}(\mathbf{x},\lambda,\nu) = f(x) + \sum_{i=1}^{m_1}\lambda_i y_i(\mathbf{x}) + \sum_{i=1}^{m_2}\nu_i h_i(\mathbf{x})$$

where $\lambda = [\lambda_1,\ldots,\lambda_{m_1}]^T$ and $\nu = [\nu_1,\ldots,\nu_{m_2}]^T$

Definition 3 (Lagrangian dual function)**.**

The Lagrange dual function $g : \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \mapsto \mathbb{R}$

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} \mathcal{L}(\mathbf{x}, \lambda, \nu)$$

Define the solution to the optimisation problem be the optimal point x^* then x^* satisfies the constraints of the optimisation problem, we have the primal feasibility conditions:

$$y_i(x^*) \leq 0, 0 \leq i \leq m_1$$
$$h_i(x^*) = 0, 0 \leq i \leq m_2$$

The optimal point minimises the Lagrangian function so we can write the dual function as

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} \mathcal{L}(\mathbf{x}, \lambda, \nu) = \mathcal{L}(\mathbf{x}^*, \lambda, \nu)$$

If $\lambda \succeq 0$, then the dual function is a lower bound for $f(x^*)$, $g(\lambda, \nu) \leq f(x^*)$ For having the dual function as a lower bound for the optimum function, the dual variables $\{\lambda_i\}_{i=1}^{m_1}$ should be non-negative. These are the dual feasibility conditions.

We can define the Lagrange dual optimisation problem for the above optimisation problem as follows

$$\max_{\lambda, \nu} g(\lambda, \nu)$$
$$\lambda \succeq 0$$

Definition 4 (Strong and weak duality)**.**
Let the solutions to the dual problem be λ^*, ν^*. For all convex and non-convex optimisation problems, the optimum dual problem is a lower bound for the optimum function $g(\lambda^*, \nu^*) \leq f(x^*)$. This is called weak duality. For some optimisation problems, we can achieve the equality, $g(\lambda^*, \nu^*) = f(x^*)$, which is called strong duality.

Assume that the problem has strong duality, the primal optimal is x^* and the dual optimal variables are λ^*, ν^* , we have

$$\begin{aligned}
f(x^*) &= g(\lambda^*, \nu^*) \\
&= \inf_{x \in \mathcal{D}} (f(x) + \sum_{i=1}^{m_1} \lambda_i^* y_i(x) + \sum_{i=1}^{m_2} \nu_i^* h_i(x)) \\
&= f(x^*) + \sum_{i=1}^{m_1} \lambda_i^* y_i(x^*) + \sum_{i=1}^{m_2} \nu_i^* h_i(x^*) \\
&= f(x^*) + \sum_{i=1}^{m_1} \lambda_i^* y_i(x^*) \\
&\leq f(x^*)
\end{aligned}$$

As we know $\lambda_i^* \geq 0$ and $y_i(x^*) \leq 0$ so we have $\lambda_i^* y_i(x^*) \leq 0$ where $0 \leq i \leq m_1$. From the above we have $f(x^*) = f(x^*) + \sum_{i=1}^{m_1} \lambda_i^* y_i(x^*) \leq f(x^*)$ which implies $\sum_{i=1}^{m_1} \lambda_i^* y_i(x^*) = 0$. We then have the

complementary slackness condition:

$$\lambda_i^* y_i(x^*) = 0, 0 \leq i \leq m_1$$

Solving the constrained optimisation problem is equivalent to minimising the Lagrangian function. The minimum can be found by setting its derivative with respect to x equal to zero, which gives the stationary condition.

Combining the primal feasibility, dual feasibility, complementary slackness, and stationarity condition, we have the Karush-Kuhn-Tucker(KKT) conditions, where the primal optimal variable x^* and the dual optimal variables λ^*, ν^* must satisfy.

Theorem 2.7.2 (Karush-Kuhn-Tucker conditions)**.**

- Stationarity condition:

$$\nabla_x \mathcal{L}(\mathbf{x}, \lambda, \nu) = \nabla_x f(x) + \sum_{i=1}^{m_1} \lambda_i^* \nabla_x y_i(x) + \sum_{i=1}^{m_2} \nu_i^* \nabla_x h_i(x) = 0$$

- Primal feasibility:

$$y_i(x^*) \leq 0, 0 \leq i \leq m_1$$
$$h_i(x^*) = 0, 0 \leq i \leq m_2$$

- Dual feasibility:

$$\lambda_i \geq 0, 0 \leq i \leq m_1$$

- Complementary slackness:

$$\lambda_i^* y_i(x^*) = 0, 0 \leq i \leq m_1$$

The optimisation problem can then be solved by the method of Lagrange multipliers.

1. Derive the Lagrangian $\mathcal{L}(\mathbf{x}, \lambda, \nu)$
2. Solve for the dual function

$$x' = \arg\min_x \mathcal{L}(\mathbf{x}, \lambda, \nu)$$
$$g(\lambda, \nu) = \mathcal{L}(x', \lambda, \nu)$$

3. Solve the dual problem to obtain the optimal dual variables

$$\lambda^*, \nu^* = \arg\max_{\lambda, \nu} g(\lambda, \nu)$$

4. Solve for the optimal primal variable

$$x^* = \arg\min_x \mathcal{L}(x, \lambda^*, \nu^*)$$

Proposition 2.7.3 (Solving NMF optimisation problem using KKT conditions)**.** The NMF optimisation problem is given as

$$\min_{W,V} \|X - WV\|_F \quad \text{subject to} \quad W \geq 0,\, V \geq 0.$$

We construct the Lagrangian $\mathcal{L}$ for the KKT conditions as follows

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}\|X - WV\|^2 - \sum_{i=1}^{n}\sum_{j=1}^{k}\lambda_{ij}w_{ij} - \sum_{i=1}^{k}\sum_{j=1}^{m}\omega_{ij}v_{ij} \\ &= \frac{1}{2}\|X - WV\|^2 - \operatorname{tr}(\Lambda W^T) - \operatorname{tr}(\Omega V^T)\end{aligned}$$

where Λ, Ω are Lagrange multiplier matrices.

The stationarity conditions require

$$\begin{aligned}\frac{1}{2}\frac{\partial\|X - WV\|^2}{\partial V_{ij}} &= \Omega_{ij} \\ \frac{1}{2}\frac{\partial\|X - WV\|^2}{\partial W_{ij}} &= \Lambda_{ij}\end{aligned}$$

Complementary slackness requires

$$\begin{aligned}\Omega_{ij}V_{ij} &= 0 \\ \Lambda_{ij}W_{ij} &= 0\end{aligned}$$

Combining the above equations we obtain

$$\begin{aligned}[W^TWV - W^TX]_{ij}V_{ij} &= 0 \\ [WVV^T - XV^T]_{ij}W_{ij} &= 0\end{aligned}$$

This leads to the update equations

$$\begin{aligned}V_{ij} &= V_{ij} \odot \frac{W^TX}{W^TWV} \\ W_{ij} &= W_{ij} \odot \frac{XV^T}{WVV^T}\end{aligned}$$

Multiplicative update rules [84] and coordinate descent [87] are used in different implementations of NMF. The multiplicative updates rules can be applied to any loss functions in the beta-divergence family

[88]. A coordinate descent algorithm is given for the loss functions of the form of Bregman divergences [87].

Multiplicative updates A multiplicative update is a form of gradient update that preserves the non-negativity of parameters. Given the gradient of a loss function, we write it as a difference of two parts

$$\nabla\mathcal{L} = [\nabla\mathcal{L}]^+ - [\nabla\mathcal{L}]^-$$

where $[\nabla\mathcal{L}]^+ > 0$ and $[\nabla\mathcal{L}]^- > 0$

The update rule for parameter Θ is given as the following form

$$\Theta \leftarrow \Theta \odot \frac{[\nabla\mathcal{L}]^-}{[\nabla\mathcal{L}]^+}$$

β-divergence [88] is a family of loss functions to measure the differences between two matrices(tensors). Commonly used error functions such as Frobenius and generalised KL divergence are special cases of β-divergence when $\beta = 2{,}1$ respectively.

Definition 5 (β-divergence)**.** Let X be the original $N \times M$ matrix and W,H be the factorised matrices of $N \times K$ and $K \times M$ respectively. The β-divergence between the original matrix and reconstruction $D(X|WH)$ are defined as follows

$$D(X|WH) = \sum_{i=1}^{N}\sum_{j=1}^{M} d(X_{ij}, WH_{ij})$$

with $d(\cdot,\cdot)$ given by

$$\begin{aligned} d(x,y) &= \frac{1}{\beta(\beta-1)}(x^\beta + (\beta-1)y^\beta - \beta x y^{(\beta-1)}) \qquad \beta \neq 0,1 \\ &= x\log\frac{x}{y} - x + y \qquad \beta = 1 \\ &= \frac{x}{y} - \log xy - 1 \qquad \beta = 0 \end{aligned}$$

Optimisation for this loss function is convex when $1 \leq \beta \leq 2$ [88].

There is a correspondence between optimisation under different matrix norms and likelihood. Minimising under the Frobenius norm corresponds to optimising under Gaussian distributions as in PCA. The KL divergence is connected to the likelihood function for Poisson distribution as it can be shown that minimising the beta divergence is equivalent to maximising the log-likelihood function.

Proof. Let $p(X|WH)$ be the likelihood function for Poisson distribution

$$\begin{aligned} \log p(X|WH) &= \sum_{i=1}^{N}\sum_{j=1}^{M}\left[v_{ij}\log\sum_{k=1}^{K} w_{ik}h_{kn} - \sum_{k=1}^{K} w_{ik}h_{kn} - \log v_{ij}!\right] \\ &= -D(X|WH) + \sum_{i=1}^{N}\sum_{j=1}^{M} v_{ij}\log v_{ij} - v_{ij} - \log v_{ij}! \\ &= -D(X|WH) + \text{const} \end{aligned}$$

□

Bregman divergences can be considered as a generalisation of β-divergence [87]. Most loss functions used in machine learning can be expressed in this form.

Definition 6 (Bregman divergence)**.** Given ϕ a convex smooth function, Bregman divergence between two non-negative matrices $X = [X_{ij}]$, $Y = [Y_{ij}]$ are defined as

$$\begin{aligned} D_\phi(X|Y) &= \sum_{i,j} D_\phi(X_{ij}|Y_{ij}) \\ &= \sum_{i,j} (\phi(X_{ij}) - \phi(Y_{ij}) + \nabla\phi(Y_{ij})(X_{ij} - Y_{ij})) \end{aligned}$$

To find non-negative matrices W, H minimising $D(X|WH)$ with respect to β-divergence, multiplicative update algorithm 3 can be used. Scalar Block coordinate descent algorithm 4 can also be derived to solve NMF with Bregman divergences. Both algorithms have a very similar functional form for the update equations. They can be considered gradient descent algorithms with different step sizes.

Algorithm 3: Multiplicative updates algorithm for NMF

Given $X \in \mathbb{R}_+^{N\times M}$, number of latent factors K, β for the beta-loss function, initialise non-negative matrices W, H randomly.
repeat
  $H \leftarrow H \frac{W^T([WH]^{\beta-2}\odot X)}{W^T[WH]^{\beta-1}}$
  $W \leftarrow W \frac{([WH]^{\beta-2}\odot X)H^T}{[WH]^{\beta-1}H^T}$
until *beta-loss function converges*;

Algorithm 4: Scalar Block coordinate descent algorithm for NMF with Bregman divergences

Given $X \in \mathbb{R}_+^{N\times M}$, number of latent factors K, ϕ function for Bregman divergences, initialise non-negative matrices $W \in \mathbb{R}_+^{N\times K}$ with columns w_k and $H \in \mathbb{R}_+^{K\times M}$ with rows h_k randomly
$X' = WH$, $E = X - X'$
repeat
  $B = \nabla^2\phi(X')$
  for $k = 1 \dots K$ **do**
    $X^k = E + w_k h_k$
    for $j = 1 \dots p$ **do**
      $h_{kj} = \left[\frac{\sum_{i=1}^n b_{ij} x_{ij}^k w_{ik}}{\sum_{i=1}^n b_{ij} w_{ik} w_{ik}}\right]_+$
    end
    for $i = 1 \dots n$ **do**
      $w_{ik} = \left[\frac{\sum_{j=1}^p b_{ij} x_{ij}^k h_{kj}}{\sum_{j=1}^p b_{ij} h_{kj} h_{kj}}\right]_+$
    end
    $E = X^k - w_k h_k$
  end
  $X' = WH$
until *Bregman divergences converges*;

NMF also has a known probabilistic interpretation which is related to Poisson factor analysis (PFA) [47]. It can be shown that under NMF, X is assumed to be a matrix of Poisson random variables with rates

given by a factorisation into two matrices Θ and Δ of Gamma distributed random variables:

$$X_{N\times M} \sim \text{Poisson}(\Theta\,\Delta) \tag{2.10}$$

$$\Theta_{N\times K} \sim \text{Gamma}(1,\infty) \tag{2.11}$$

$$\Delta_{K\times M} \sim \text{Gamma}(1,\infty) \tag{2.12}$$

Θ and Δ correspond, respectively, to cell and gene scores onto the K latent factors. Note that Poisson-Gamma random variables are related to the Negative Binomial distribution, which is widely used to model scRNA-seq counts.

Variational Inference

(Stochastic) variational inference has been used in a wide range of generative models including Latent Dirichlet allocation (LDA), Hierarchical Poisson Factorisation (HPF) [48] and variational autoencoders (VAE) [89]. It is the mathematical foundation to derive the update equations used in integrative Hierarchical Poisson Factorisation (IHPF).

In the following, we develop the variational inference algorithm for inferring the class of models where the complete conditionals are in the exponential family following the paper by Hoffman et al.[90].

Consider a model with N observations $x = x_{1:N}$ with global hidden variable β, N local hidden variable $z = z_{1:N}$ where each of them is a collection of J variables $z_n = z_{n,1:J}$, hyper-parameter α. The joint density is given by

$$p(x, z, \beta|\alpha) = p(\beta|\alpha)\prod_{i=1}^{N} p(x_n, z_n|\beta)$$

Assume the complete conditionals in the model belongs to the exponential family,

$$p(\beta|x, z, \alpha) = h(\beta)\exp[\eta_g(x, z, \alpha)^T t(\beta) - a_g(\eta_g(x, z, \alpha))]$$
$$p(z_{nj}|x_n, z_{n,-j}, \beta) = h(z_{nj})\exp[\eta_l(x_n, z_{n,-j}, \beta)^T t(z_{nj}) - a_l(x_n, z_{n,-j}, \beta)]$$

where $h(\cdot)$ and $a(\cdot)$ are the base measure and log-normalizer of the exponential family with $\eta(\cdot)$ and $t(\cdot)$ being the natural parameter and sufficient statistics.

Variational inference proposes a variational family distribution $q(z, \beta)$ to approximate the intractable posterior distribution $p(z, \beta|x)$. It aims to minimise the Kullback-Leibler (KL) divergence between the variational distribution to the posterior distribution. It can be done by maximising the evidence lower bound, a lower bound on the logarithm of the marginal density $\log p(x)$ as shown in Proposition 2.7.5.

Proposition 2.7.4 (Deriving evidence lower bound)**.**

$$\begin{aligned}
\log p(x) &= \log \int p(x, z, \beta) dz d\beta \\
&= \log \int p(x, z, \beta)\frac{q(z, \beta)}{q(z, \beta)} dz d\beta \\
&= \log\left(\mathbb{E}_q\left[\frac{p(x, z, \beta)}{q(z, \beta)}\right]\right) \\
&\geq \mathbb{E}_q[\log p(x, z, \beta)] - \mathbb{E}_q[\log q(z, \beta)] \\
&\triangleq ELBO(q)
\end{aligned}$$

Proposition 2.7.5. Evidence lower bound is equal to the negative KL divergence up to a constant

$$\begin{aligned} ELBO(q) &\triangleq \mathbb{E}_q[\log p(x, z, \beta)] - \mathbb{E}_q[\log q(z, \beta)] \\ &= \mathbb{E}_q[\log p(z, \beta|x)] + \mathbb{E}_q[\log p(x)] - \mathbb{E}_q[\log q(z, \beta)] \\ &= \log p(x) - \mathbb{KL}(q(z, \beta)||p(z, \beta|x)) \end{aligned}$$

The variational distribution is usually set as the mean-field family which assumes the factorisation of hidden variables to be in the same exponential family as the complete conditionals.

Definition 7 (Mean-field variational family)**.** Given $h(\cdot)$ and $a(\cdot)$ the base measure and log-normalizer of the exponential family defined for the complete conditionals, the mean-field variational family is given as

$$\begin{aligned} q(z, \beta) &= q(\beta|\lambda) \prod_{n=1}^{N} \prod_{j=1}^{J} q(z_{nj}|\phi_{nj}) \\ q(\beta|\lambda) &= h(\beta) \exp[\lambda^T t(\beta) - a_g(\lambda)] \\ q(z_{nj}|\phi_{nj}) &= h(z_{nj}) \exp[\phi_{nj}^T t(z_{nj}) - a_l(\phi_{nj})] \end{aligned}$$

where λ are the global parameters, ϕ_n are the local parameters

A coordinate ascent algorithm is used to optimise the evidence lower bound. Each variational parameter is updated iteratively while holding others fixed. Under the above assumptions on the model and variational distribution, the coordinate update can be obtained in closed form. Each variational parameter is set to the expected natural parameter of its complete conditional.

Algorithm 5: Coordinate Ascent Variational inference

Initialise: Global variational parameters λ^0
while *the ELBO has not converged* **do**
  for *each local variational parameters* ϕ_{nj} **do**
    Set $\phi_{nj}^t = \mathbb{E}_{q^{t-1}}[\eta_l(x_n, z_{n,-j}, \beta)]$
  end
  Update global variational parameters $\lambda^t = \mathbb{E}_{q^t}[\eta_g(x, z, \alpha)]$
end

The coordinate update for the global parameter λ is derived in detail as follows. The coordinate update for the local parameters ϕ_{nj} can be similarly derived.

Proof. The expected log joint distribution can be rewritten as

$$\mathbb{E}_q[\log p(x, z, \beta)] = \mathbb{E}_q[\log p(x, z)] + \mathbb{E}_q[\log p(\beta|x, z)]$$

The evidence lower bound written as a function of λ becomes

$$ELBO(\lambda) = \mathbb{E}_q[\log p(\beta|x, z)] - \mathbb{E}_q[\log q(\beta)] + \text{ const}$$

Using properties of the exponential family where the expectation of sufficient statistics is the gradient of the log normaliser and substituting the form of the complete conditionals, we obtain

$$ELBO(\lambda) = \mathbb{E}_q[\eta_g(x, z, \alpha)]^T \nabla_\lambda a_g(\lambda) - \lambda^T \nabla_\lambda a_g(\lambda) + a_g(\lambda) + \text{ const}$$

Take the gradient,

$$\nabla_\lambda ELBO(\lambda) = \nabla^2_\lambda a_g(\lambda)(\mathbb{E}_q[\eta_g(x, z, \alpha)] - \lambda)$$

Setting the gradient to zero gives the update equation

$$\lambda = \mathbb{E}_q[\eta_g(x, z, \alpha)]$$

□

The convergence of ELBO can be improved by using the natural gradient instead of the usual (Euclidean) gradient. Natural gradient considers the information geometry of the parameter space with the use of a Riemannian metric [91].

Definition 8 (Natural gradient)**.** Difference between two probability distributions can be measured by symmetrised KL divergence

$$D^{sym}_{KL}(\lambda, \lambda') = \mathbb{E}_\lambda \left[\log \frac{q(\beta|\lambda)}{q(\beta|\lambda')}\right] + \mathbb{E}_{\lambda'} \left[\log \frac{q(\beta|\lambda')}{q(\beta|\lambda)}\right]$$

A Riemannian metric $G(\lambda)$ is used to define linear transformations of λ where the squared Euclidean distance between λ and $\lambda + d\lambda$ is the KL between two distributions $q(\beta|\lambda)$ and $q(\beta|\lambda + d\lambda)$

$$d\lambda^T G(\lambda) d\lambda = D^{sym}_{KL}(\lambda, \lambda + d\lambda)$$

Let G to be the Fisher information matrix of $q(\beta|\lambda)$, the natural gradient $\hat{\nabla}_\lambda f(\lambda)$ is defined as

$$\begin{aligned} G(\lambda) &= \mathbb{E}_\lambda[(\nabla_\lambda \log q(\beta|\lambda))(\nabla_\lambda \log q(\beta|\lambda))^T] \\ \hat{\nabla}_\lambda f(\lambda) &\triangleq G(\lambda)^{-1} \nabla_\lambda f(\lambda) \end{aligned}$$

Under the assumption of $q(\beta|\lambda)$ in the exponential family, the metric is the second derivative of the log normaliser.

$$\begin{aligned} G(\lambda) &= \mathbb{E}_\lambda[(\nabla_\lambda \log q(\beta|\lambda))(\nabla_\lambda \log q(\beta|\lambda))^T] \\ &= \mathbb{E}_\lambda[(t(\beta) - \mathbb{E}_\lambda(t(\beta)))(t(\beta) - \mathbb{E}_\lambda(t(\beta)))^T] \\ &= \nabla^2_\lambda a_g(\lambda) \end{aligned}$$

The natural gradient of the evidence lower bound with respect to the global and local variables is then given by

$$\begin{aligned} \hat{\nabla}_\lambda ELBO &= \mathbb{E}_\phi[\eta_g(x, z, \alpha)] - \lambda \\ \hat{\nabla}_{\phi_{nj}} ELBO &= \mathbb{E}_{\lambda, \phi_{n,-j}}[\eta_l(x_n, z_{n,-j}, \beta)] - \phi_{n,-j} \end{aligned}$$

Stochastic Variational inference can be used to speed up the calculation of the natural gradient. A

noisy gradient is calculated on a subset of data instead of the whole data as in stochastic gradient descent.

Algorithm 6 gives the stochastic variational inference algorithm for learning the variational parameters under the assumptions above.

Algorithm 6: Stochastic Variational inference

Initialise: Global variational parameters λ^0

Set the step-size schedule ρ_t. For example, $\rho_t = (t+\tau)^{-\kappa}$ with the forgetting rate $kappa \in (0.5, 1]$ and delay $\tau \geq 0$ down-weights early iterations.

while *the ELBO has not converged* **do**

Sample a data point x_i uniformly from the dataset.

Denote (x_i^N, z_i^N) to be the dataset formed by N replicates of the observation and hidden variables

Compute local variational parameter

$$\phi = \mathbb{E}_{q^{t-1}}[\eta_l(x_i^N, z_i^N)]$$

Compute intermediate global parameters as though x_i is replicated N times

$$\hat{\lambda} = \mathbb{E}_{q^t}[\eta_g(x_i^N, z_i^N)]$$

Update current estimate of global variational parameters

$$\lambda^t = (1-\rho_t)\lambda^{t-1} + \rho_t\hat{\lambda}$$

end

Hierarchical Poisson Factorisation

Hierarchical Poisson Factorisation (HPF) directly builds on Poisson factorisation where an additional layer of gamma variables is introduced to model the gene and cell capacity. The generative model is given in Algorithm 7 [48]. The model presented in the following assembles the ones presented for IHPF in chapter 2 as HPF can be considered as a special case of IHPF with a single batch and a noise ratio of 0. For completeness, I also present the model and update equations for HPF from Blei et al.[48].

Algorithm 7: single-cell hierarchical Poisson factorisation

Result: Expression level for discrete scRNA-seq expression matrix

Set hyper-parameters a, a', b', c, c', d' ;

for *each cell i* **do**

  Sample capacity $\xi_i \sim Gamma(a', b')$;

  for *each factor k* **do**

    Sample weight $\theta_{i,k} \sim Gamma(a, \xi_i)$;

  end

end

for *each gene g* **do**

  Sample capacity $\eta_g \sim Gamma(c', d')$;

  for *each factor k* **do**

    Sample weight $\beta_{g,k} \sim Gamma(c, \eta_g)$;

  end

end

for *each cell i and gene g* **do**

  Sample observed expression level $x_{i,g} \sim Poisson(\theta_i \beta_g^T)$;

end

The posterior distribution is analytically intractable so variational inference methods are used to find an approximation to the true posterior. In this project mean-field, variational family is used to approximating the posterior. The coordinate Ascent algorithm is used to fit the parameters of the mean-field variational family.

For HPF, a conditionally conjugate version of the model with an additional layer of latent variables is recommended by Levitin et. al. [48]. For each cell i and gene g, K latent variables are added. $z_{igk} \sim Poisson(\theta_{ik}\beta_{gk})$ such that $x_{ig} = \sum_k z_{igk}$. The marginal distribution of observed molecular counts is preserved as the sum of independent Poisson random variables is a Poisson random variable.

A mean-field variational family over the latent variables is proposed with variational parameters to have the same form as complete conditionals. Denote $\mathbf{z}$ as the vector of latent variables z_k, $\gamma_{ik}, \lambda_{gk}, \kappa_i, \tau_g$ are Gamma distribution with different shapes and scales. ϕ_{ig} is a multinomial.

$$q(\theta, \beta, \xi, \eta, \mathbf{z}) = \prod_{ik} q(\theta_{ik}|\gamma_{ik}) \prod_{gk} q(\beta_{gk}|\lambda_{gk}) \prod_{i} q(\xi_i|\kappa_i) \prod_{g} q(\eta_g|\tau_g) \prod_{ig} q(\mathbf{z}_{ig}|\phi_{ig})$$

Algorithm 8 implements the coordinate ascent algorithm for the HPF model. The gene weights, cell weights and expression levels are updated in order during each iteration until the algorithms converge [48].

Probabilistic Count Matrix Factorisation

Probabilistic Count Matrix Factorisation (pCMF) [76] extends NMF by modelling over-dispersion, dropouts and gene selection. Over-dispersion in counts is modelled by a negative binomial distribution expressed as a compound Gamma-Poisson distribution. Dropout events are modelled with gene-specific dropout rates π_p^D through the indicator variable D_{np}. To introduce sparsity in gene selection we introduce indicator variable S_{kp} to model whether gene p contributes to cell scores θ_{kp} with prior probability π_p^S.

The variational EM algorithm is used to jointly infer the posterior distribution and estimate the hyper-parameters. A mean-field variational family is learnt which minimises the KL divergence between the

Algorithm 8: Coordinate Ascent Algorithm for likelihood inference in HPF

Result: variational distribution approximation to true posterior

Set shape parameters of the gene and cell capacities, where K is the number of factors ;

$$\kappa_i^{shp} = a' + Ka$$
$$\tau_g^{shp} = c' + Kc$$

repeat

  for *each gene g* **do**

    Set the gene weights and capacity

$$\lambda_{gk}^{shp} = c + \sum_i x_{ig}\phi_{igk}$$
$$\lambda_{gk}^{rte} = \frac{\tau_g^{shp}}{\tau_g^{rte}} + \sum_i \frac{\gamma_{ik}^{shp}}{\gamma_{ik}^{rte}}$$
$$\tau_g^{rte} = d' + \sum_k \frac{\lambda_{gk}^{shp}}{\lambda_{gk}^{rte}}$$

  end

  for *each cell i* **do**

    Set the cell weights and capacity

$$\gamma_{ik}^{shp} = a + \sum_g x_{ig}\phi_{igk}$$
$$\gamma_{ik}^{rte} = \frac{\kappa_i^{shp}}{\kappa_i^{rte}} + \sum_g \frac{\lambda_{gk}^{shp}}{\lambda_{ik}^{rte}}$$
$$\kappa_i^{rte} = c' + \sum_k \frac{\gamma_{ik}^{shp}}{\gamma_{ik}^{rte}}$$

  end

  for *each cell i and gene g such that* $x_{i,g} > 0$ *;* **do**

    set the multinomial

$$\phi_{ig} \propto \exp(\Psi(\gamma_{ik}^{shp}) - \log \gamma_{ik}^{rte} + \Psi(\lambda_{gk}^{shp}) - \log \lambda_{gk}^{rte})$$

  end

until *change to marginal log likelihood* $\leq$ *0.001%*;

$$
\begin{aligned}
x_{np} &= \sum_{k=1}^{K} x_{npk} \\
x_{npk} &\sim (1 - D_{np}) \times \delta_0 + D_{np} \times \text{Poisson}(\phi_{nk}\theta_{kp}) \\
\phi_{nk} &\sim \text{Gamma}(a_k, b_k) \\
\theta_{kp} &\sim (1 - S_{kp}) \times \delta_0 + S_{kp} \times \text{Gamma}(c_k, d_k) \\
D_{np} &\sim \text{Bernoulli}(\pi_p^D) \\
S_{kp} &\sim \text{Bernoulli}(\pi_p^S)
\end{aligned}
$$

Figure 2.13: pCMF model

true posterior distribution and the variational distribution. The E-step of the EM algorithm corresponds to calculating the approximated expectation of joint likelihood and the M-step corresponds to maximising the joint likelihood with respect to the hyperparameters for the gamma prior and sparsity rate for counts and genes.

pCMF address the issue of sparsity in scRNA-seq datasets by introducing two boolean matrix to simulate the dropout effects on the counts and gene selection. Extra hyper-parameters are introduced which increases model complexity. The computational cost for running pCMF is much higher than traditional methods such as PCA. It is less effective in detecting rare cell types and trajectory inference than PCA and NMF in a recent review of dimension reduction methods for scRNA-seq datasets [40].

Hierarchical gamma-negative binomial

Hierarchical gamma-negative binomial (hGNB) model is used to analyse scRNA-seq data [92].

The generative model is given in Figure 2.14a. Given $N \times P$ count matrix Y with cell covariates $x_n \in \mathbb{R}^m$ and gene covariates $z_p \in \mathbb{R}^q$, we learn cell dispersion r_n with a gamma prior which accounts variability between cells. Regression coefficients β_p, δ_n are learnt to account for batch effects in cells or the effect of gene length and GC-content in genes. $\phi_p^T \theta_n$ represents the factorisation of counts after correcting for various batch effects.

Negative binomial distribution can be expressed as a Poisson distribution with rate following a suitable gamma distribution. The model above can be rewritten as in Figure 2.14b.

hGNB is different from other PFA-based methods such as HPF on several aspects. The Poisson rate for counts is not expressed as a product of two lower dimension matrices of gamma prior. Factorisation is performed on the shape parameter only with cell and gene covariates effects corrected similar to Bayesian General Linear models (GLM).

A major limitation of hGNB is that it requires cell and gene metadata to model the technical effects in the datasets. The model is also more complicated than others as a result. Inference of hGNB is performed by Gibbs Sampling with data augmentation through Polya-Gamma distribution to sample regression coefficients. The long computational time required excludes this method for data integration.

Compare different matrix factorisation methods

We present the graphical representations of the above models in Figure 2.15. HPF shows a simple and robust design while other models are over-complicated with additional variables and hyper-parameters that

$$
\begin{aligned}
y_{np} &\sim \text{Negative Binomial}(r_n, s_{np}) \\
r_n &\sim \text{Gamma}(e_0, 1/h) \\
\text{logit}(s_{np}) &= \beta_p^T x_n + \delta_n^T z_p + \phi_p^T \theta_n \\
\beta_p &\sim \prod_{m=1}^{M} \text{Normal}(\beta_{pm}, 0, \alpha_m^{-1}) \\
\delta_n &\sim \prod_{q=1}^{Q} \text{Normal}(\delta_{jq}, 0, \eta_q^{-1}) \\
\phi_p &\sim \text{Normal}(\phi_p, 0, I_K) \\
\theta_n &\sim \prod_{k=1}^{K} \text{Normal}(\theta_{nk}, 0, \gamma_k^{-1}) \\
\alpha_m, \eta_q, \gamma_k &\sim \text{Gamma}(e_0, 1/f_0)
\end{aligned}
$$

(a) hGNB model

$$
\begin{aligned}
y_{np} &\sim \text{Poisson}(\lambda_{np}) \\
\lambda_{np} &\sim \text{Gamma}(r_n, \phi_{pn}) \\
r_n &\sim \text{Gamma}(e_0, 1/h) \\
\phi_{pn} &= \exp(\beta_p^T x_n + \delta_n^T z_p + \phi_p^T \theta_n) \\
\beta_p &\sim \prod_{m=1}^{M} \text{Normal}(\beta_{pm}, 0, \alpha_m^{-1}) \\
\delta_n &\sim \prod_{q=1}^{Q} \text{Normal}(\delta_{jq}, 0, \eta_q^{-1}) \\
\phi_p &\sim \text{Normal}(\phi_p, 0, I_K) \\
\theta_n &\sim \prod_{k=1}^{K} \text{Normal}(\theta_{nk}, 0, \gamma_k^{-1}) \\
\alpha_m, \eta_q, \gamma_k &\sim \text{Gamma}(e_0, 1/f_0)
\end{aligned}
$$

(b) hGNB model written under PFA framework

Figure 2.14: hGNB generative models

make training difficult. Other methods have different limitations that preclude them to be interpretable and robust. PCA, ZIFA and ZINB-WaVE learn the parameters in a logarithmic scale such that the latent space is not interpretable. ZINB-WavE cannot be applied on large datasets due to computational time constraints as shown in various studies [40, 48]. pCMF assumes a component that randomly creates zero observation ignoring the actual underlying expression in cells [76]. The model implicitly assumes a systematic effect to drop all molecules from a particular gene at a particular cell. hGNB attempts to model the excess zeros in the datasets through zero-inflated distributions and correct the batch differences with cell and gene covariates. Newer perspectives suggest Poisson distribution is sufficient for modelling the sparsity in count matrices and discourages the use of zero-inflated distribution which introduces over-fitting of model [75]. For the above reasons, we do not use ZINB-WaVE, pCMF and hGNB to run the computations in the thesis.

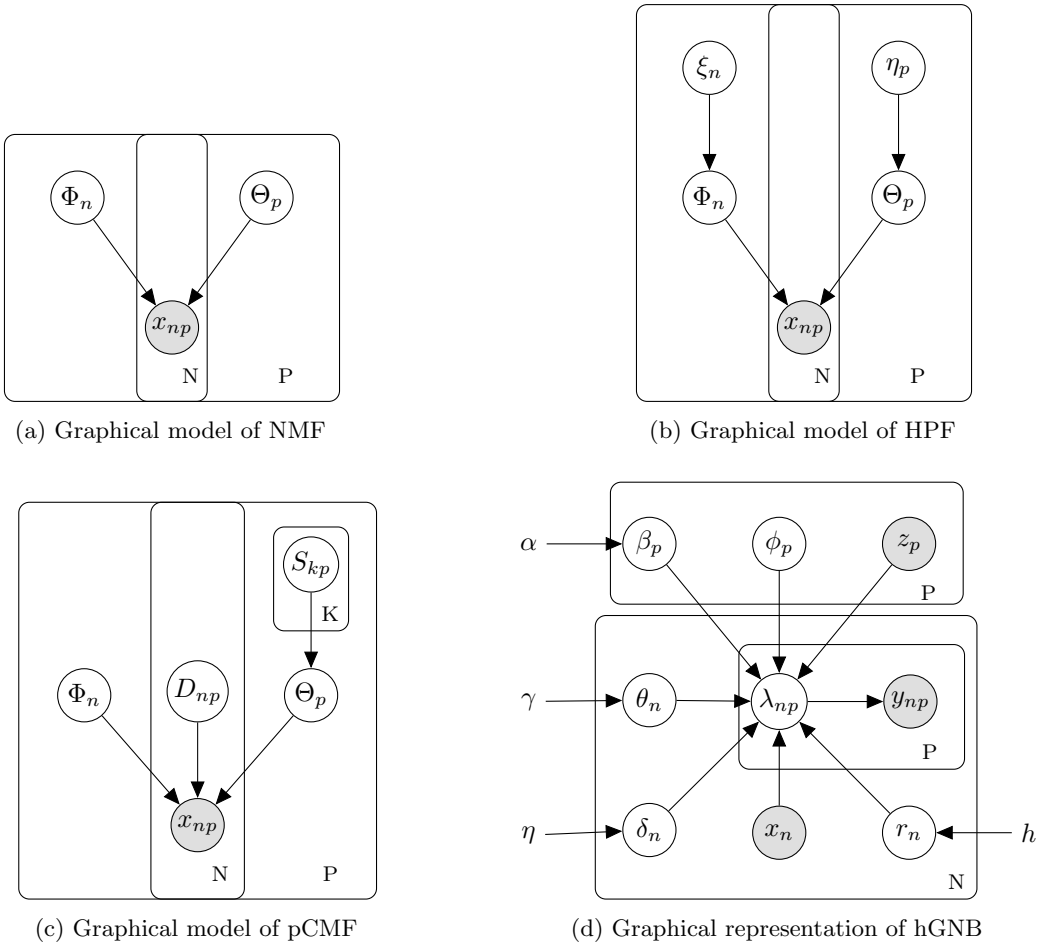
Variational Autoencoders

Mean-field variational inference assumes the expectations with respect to approximate posterior be tractable, limiting its usage to a few simple classes of posterior functions. Auto-encoding variational Bayesian (AEVB) [81] is proposed to obtain an unbiased estimator of the Evidence Lower bound (ELBO), without the use of time-consuming methods such as MCMC.

Consider a dataset $\mathbf{X} = \{x^i\}_{i=1}^N$ of N i.i.d. samples generated as follows. 1. From a prior distribution $p_{\theta_*}(z)$ we generate a latent variable z^i. 2. With likelihood $p_{\theta_*}(x|z)$ we generate the data x^i. The prior $p_{\theta_*}(z)$ and likelihood $p_{\theta_*}(x|z)$ belongs some parametric families of $p_\theta(z)$ and $p_\theta(x|z)$.

Introduce an approximate model $q_\phi(z|x)$ the approximation to the intractable true posterior $p_\theta(z|x)$. Unlike mean-field variational inference, the parameters ϕ do not have to follow from a closed-form expectation. AEVB can be used to jointly infer the generative model parameters θ and approximate model parameters ϕ.

The marginal likelihood of data can be written as the sum of KL divergence between the true and



(a) Graphical model of NMF

(b) Graphical model of HPF

(c) Graphical model of pCMF

(d) Graphical representation of hGNB

Figure 2.15: Graphical representations of different matrix factorisation methods. Coloured circles represent observations while the rest represent random variables. Each model assumes there are N cells with P genes. NMF: Non-negative matrix factorisation HPF: Hierarchical Poisson factorisation pCMF: probabilistic Count Matrix Factorization. hGNB: Hierarchical Gamma Negative Binomial

approximate posterior and the evidence lower bound.

$$\begin{aligned}\log p_\theta(x^i) &= \mathbb{D}_{KL}(q_\phi(z|x^i)||p_\theta(z|x^i)) + \mathbb{E}_{q_\phi(z|x)}(-\log q_\phi(z|x) + \log p_\theta(z,x)) \\ &\triangleq \mathbb{D}_{KL}(q_\phi(z|x^i)||p_\theta(z|x^i)) + \mathcal{L}(\theta,\phi,x^i)\end{aligned}$$

Computing the gradient of the lower bound w.r.t. ϕ using the usual Monte Carlo estimator of the form $\nabla_\phi \mathbb{E}_{q_\phi(z)}(f(z))$ gives a poor performance as it has a high variance.

Assume for the approximate posterior $q_\phi(z|x)$ the random variable $\tilde{z} \sim q_\phi(z|x)$ can be reparameterised as $\tilde{z} = g_\phi(\epsilon, x)$ where $\epsilon \sim p(\epsilon)$ follows some auxiliary noise distribution with $g_\phi(\epsilon, x)$ a differentiable transformation. This reparameterisation trick works for a large family of distributions, including Gaussian distributions which are commonly used to represent to approximate posterior.

Using this reparameterisation trick, the stochastic gradient variational Bayes estimator of the lower

bound can be written as

$$\begin{aligned}\mathcal{L}(\theta, \phi, x^i) &= \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x^i, z^{i,l}) - \log q_\phi(z^{i,l}|x^i) \\ &= -\mathbf{D}_{KL}(q_\phi(z|x^i)||p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x^i|z^{i,l})\end{aligned}$$

where $z^{i,l} = g_\phi(\epsilon^{i,l}, x^i)$ and $\epsilon^l \sim p(\epsilon)$

Algorithm 9: Auto-Encoding Variational Bayes(AEVB) using minibatch SGD

Initialise: generative model parameters θ, recognition model parameters ϕ
while *parameters* θ, ϕ *not converged* **do**
Sample $\mathbf{X}^M = \{x^i\}_{i=1}^M$ a minibatch of M data points from the dataset
Sample ϵ from the noise distribution $p(\epsilon)$
Calculate evidence lower bound $\mathcal{L}(\theta, \phi, \mathbf{X}^M)$

$$\begin{aligned}\mathcal{L}(\theta, \phi, \mathbf{X}^M) &= \frac{N}{M}\sum_{i=1}^{M} \mathcal{L}(\theta, \phi, x^i) \\ &= \frac{N}{M}\sum_{i=1}^{M} -\mathbf{D}_{KL}(q_\phi(z|x^i)||p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x^i|z^{i,l})\end{aligned}$$

where $z^{i,l} = g_\phi(\epsilon^{i,l}, x^i)$ and $\epsilon^l \sim p(\epsilon)$
Update parameters θ, ϕ using the gradient of evidence lower bound by SGD or other methods
end

A simple model of Variational Autoencoders (VAE) can be obtained using AEVB by setting the likelihood $p_\theta(x|z)$ to be Gaussian (for continuous data) or Poisson (for discrete data) with parameters given by the decoding MLP and the approximate posterior $q_\phi(z|x)$ to be a multivariate Gaussian with diagonal covariance structure with mean and variance the output from the encoding MLP.

Different implementations of Variational Autoencoders for single-cell RNA-seq analysis can be obtained by setting the likelihood to follow different distributions such as Negative Binomial or Zero-Inflated Poisson and the approximate posterior to follow different distributions such as mixture Gaussian.

In scVI[38], each expression value x_{ng} is drawn independently through the following process.

$$\begin{aligned}z_n &\sim \text{Normal}(0, 1) \\ l_n &\sim \text{Log Normal}(l_\mu, l_\sigma^2) \\ \rho_n &\sim f_w(z_n, s_n) \\ w_{ng} &\sim \text{Gamma}(\rho_n^g, \theta) \\ y_{ng} &\sim \text{Poisson}(l_n w_{ng}) \\ h_{ng} &\sim \text{Bernoulli}(f_h^g(z_n, s_n)) \\ x_{ng} &= y_{ng} \text{ if } h_{ng} = 0 \\ &= 0 \text{ otherwise}\end{aligned}$$

Denote the number of batches be B, then $l_\mu, l_\sigma \in \mathbb{R}_+^B$ parameterise the prior for the scaling factor (on a log scale). They are set to be the empirical mean and variance of the log-library size per batch.

f_w, f_h are neural networks that map the latent space and batch annotation to the gene space. Softmax activation is used in the output layer of f_w to encode the mean proportion of transcripts expressed across all genes. Fully connected layers are used in the neural networks with dropout regularisation and batch normalisation.

We can integrate the latent variables so that $p(x_{ng}|z_n, s_n, l_n)$ is a zero-inflated negative binomial distribution with mean $l_n\rho_n^g$, gene specific dispersion θ^g and zero-inflated probability $f_h^g(z_n, s_n)$

Variational inference is then used to approximate the posterior $p(z_n, l_n|x_n, s_n)$ with $q(z_n, l_n|x_n, s_n)$, which is given by mean-field $q(z_n, l_n|x_n, s_n) = q(z_n|x_n, s_n)q(l_n|x_n, s_n)$. The variational distribution $q(z_n|x_n, s_n)$ is chosen to be Gaussian with a diagonal covariance matrix, where its mean, and covariance are given by an encoder network. Similarly, variational distribution $q(l_n|x_n, s_n)$ is chosen to be log-normal with the scalar mean and variance is also given by an encoder network.

2.7.2 Mathematical background of data integration tools in scRNA-seq datasets

In this section, we present the mathematical foundations and inherent modelling assumptions of different data integration methods/tools for scRNA-seq datasets. Most of the tools can be considered as an extension of the dimensionality reduction methods presented in section 2.7.1 while sharing very similar mathematical properties.

BBKNN

BBKNN is an extremely efficient graph-based data integration algorithm for scRNA-seq datasets [49]. BBKNN builds a k-nearest neighbours graph on the latent cell space that is balanced across all batches of data. The method relies on the assumption that are cell types that can be found across batches and the differences in genomics measurement for cells of the same type across batches caused by batch effects are less than the biological differences between different cell types within a batch. This assumption is shared by all graph-based data integration algorithms.

Scanorama

Nearest Neighbour methods are used by Scanorama [39] and other data integration tools to combine latent spaces obtained from dimension reduction methods. Different tools will have slightly different ways to score the matched pair of cells (anchors) across datasets. Scanorama uses the Gaussian kernel to construct the weight matrices for matched cells.

Given latent spaces $W_1, \dots W_d$ for datasets $D_1, \dots D_d$ to be integrated, nearest neighbours are found for each cell with cells from other datasets. The results are denoted as N_i which is a collection of cell pairs (x, y) where $x \in D_i$, $y \in D_j$, $j \neq i$ with y a k-nearest neighbour of x. Mutual links between two datasets D_i, D_j are then identified as

$$M_{ij} = M_{ji} = \{(x_i, x_j) : (x_i, x_j) \in N_i \wedge (x_j, x_i) \in N_j\}$$

Scanorama uses an approximate algorithm that combines hyperplane locality sensitive hashing (LSH) and random projection trees for nearest neighbour search.

Each pair of dataset are then ranked by the ratio of matched cells r_{ij} to determine the order of building paranoma. $r_{ij} = max(\frac{n_{ij}}{n_i}, \frac{n_{ji}}{n_j})$ where n_i, n_j are the number of cells in dataset D_i, D_j, $n_{ij} = |\{x_i : x_i \in D_i, x_i \in M_{ij}\}|$, $n_{ji} = |\{x_j : x_j \in D_j, x_j \in M_{ij}\}|$

Batch correction is done by subtracting the weighted difference between the dataset to be corrected and the reference dataset. Let D_i to be the dataset to be corrected with D_j the reference with count matrices E_i, E_j respectively. Denote E_i', E_j' to be count matrices with rows corresponding to pairs of cells in M_{ij}, the weight matrix $\Gamma_i \in \mathbb{R}^{n_i \times |M_{ij}|}$ calculates the weight between each cell in D_i with the matched cells in D_i by a Gaussian kernel. σ is fixed to be a constant of 15.

$$[\Gamma_i]_{xy} = \exp\left(-\frac{\sigma}{2}\|[E_i]_{x,:} - [E_i']_{y,:}\|_2^2\right)$$

The correction for any cell x in dataset D_i can be performed by

$$v_x = \frac{[\Gamma_i]_{x,:}(E'_i - E'_j)}{\sum_{y \in M_{ij}} \Gamma_i]_{x,y}}$$
$$[E_i]_{x,:} \leftarrow [E_i]_{x,:} - v_x$$

The MNN algorithm used by Scanorama and other tools are dimension reduction method agnostic. They can be applied on latent spaces obtained from any suitable dimension reduction methods (with normalisation). The default dimension reduction model used by Scanorama is PCA. The latent space obtained are not sparse and interpretable, preventing direct integration of cells. MNN or other time-consuming algorithm is then needed to recover the connections and structure in the datasets.

Non-negative matrix factorisation produces latent space that are often directly interpretable. Clustering by cell types can often be easily obtained from the block structure of latent space. Batch correction can then performed for each cluster without the need to identify nearest neighbours for each cell. We will demonstrate this speeds up calculation without sacrificing much performance.

Integrating data sets obtained with different technologies under different experimental settings poses additional challenges when we want to capture biologically meaningful signals in feature (gene) space when there are technical batch effects in the sample (cell) space. Traditionally, integrative PCA (IPCA) was developed for batch correction in bulk RNA transcriptomics data, but it does not perform satisfactorily for scRNA-seq data due to the high sparsity in this type of data [33].

Integrative NMF

Integrative NMF (INMF) was introduced as an extension of NMF to harmonise data sets that share their feature space [50]. Let us consider S batches (i.e., data sets collected from different sources) with non-negative data matrices $X_s \in \mathbb{R}_+^{N_s \times M}$, $s = 1, \ldots, S$, so that $N = \sum_{s=1}^{S} N_s$.

As above, we want to find a description of the data set in terms of K latent factors, by obtaining non-negative matrices $W \in \mathbb{R}_+^{N \times K}$, $V_s \in \mathbb{R}_+^{K \times M}$, $s = 1, \ldots, S$, and $B \in \mathbb{R}_+^{K \times M}$ containing, respectively, the cell scores, the gene scores specific to each data set, and the gene scores common to all the data sets. The INMF factorisation is achieved by minimising the following loss function:

$$\sum_{s=1}^{S} \|X_s - W(\mathbf{1}_s, :)(V_s + B)\|_F + \frac{1}{\alpha} \sum_{s=1}^{S} \|W(\mathbf{1}_s, :)V_s\|_F \ . \tag{2.13}$$

where $\mathbf{1}_s$ is an indicator function for the cells in batch s and therefore $W(\mathbf{1}_s, :) \in \mathbb{R}_+^{N_s \times K}$ is the submatrix containing the cell scores of cells in batch s. The hyper-parameter α, which we denote as the *noise ratio*, regularises the magnitude of the components of the different data sets.

Chapter 3

Online learning techniques for prediction of temporal tabular datasets with regime changes

3.1 Introduction

As investors explore new ways to generate profit, machine learning (ML) models are increasingly used as part of trading strategies, e.g., to predict the future return of stocks or stock portfolios. In particular, deep learning models for time-series data, such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), have been applied to the prediction of future stock returns [93, 94, 95]. However, a major challenge for such methods is the highly stochastic, non-stationary and non-ergodic nature of financial data, which violates the assumptions of many algorithms. Furthermore, deep learning models are over-parameterised, with numbers of parameters orders of magnitude larger than typical sizes of time series data. Therefore, deep models can be easily overfitted to specific patterns in historical market data not present in future market data, and the overfitting worsens with more complex neural network architectures, such as Long Short Term Memory (LSTM) or Transformer networks. In addition, the continuous influx of data, coupled with possible regime changes, requires costly updating and retraining of such models. As a consequence, such methods can lack reproducibility and robustness for the prediction of future market data.

As pointed out in recent reviews [96, 97, 30, 29, 17], replication of ML studies is often difficult due to several issues, including data leakage [97], program bugs [98], data and code usability [99], model representation and evaluation [96]. These problems are currently hindering the usage of ML in high-risk decision processes, such as healthcare and finance. For financial trading applications in particular, these issues can have critical effects on the validity of results. Data leakage, in the form of look-ahead bias or overlap of training/test sets [100], can inflate in-sample performance with poor performance when deployed live. Furthermore, black-box ML models, such as neural networks, can lack robustness as they are highly sensitive to small changes in parameters, random seeds and data, thus resulting in highly variable predictions. The non-stationarity of data and the presence of regime changes also mean that ML models need to be re-trained with the latest financial data, a task that is not only computationally costly but also introduces further uncertainty. Yet most studies do not consider model performance when trained

on different segments of historical market data [93, 94, 95, 101, 102]. Although reinforcement learning (RL) in online learning settings allows ML models to adapt to changing environments, deep reinforcement learning models are complex and require large computational resources [103]. Indeed, applying RL to stock trading is challenging since the complexity of the action space increases exponentially with the number of stocks in the portfolio.

The above issues suggest the need to develop robust ML pipelines for trading applications possibly based on simpler models that can still operate on non-stationary, highly stochastic data under regime changes. Here we consider such a pipeline for *temporal tabular data*, which allows the use of traditional ML models, such as Gradient Boosting Decision Trees (GBDTs) and other ensemble methods, to trade stocks [104]. In particular, we find that Gradient Boosting models, which are known to be robust to data perturbations, outperform neural network models in our tasks. We also show that improved robustness of ML models and adaptation to regime changes can be attained by employing: (i) dynamic feature projection, a simple approach that reduces the linear correlation to a subset of features evolving in time, and (ii) dynamic model selection, a simple optimisation procedure that selects optimal models from an ensemble based on recent performance. These approaches robustly improve trading performances by reducing volatility and drawdown during adversarial market regimes.

To exemplify the above issues, we consider a benchmark financial data platform that is continuously updated and curated under the Numerai tournament of stock portfolio prediction [105]. Numerai is a hedge fund that organises a data science competition (as of May 2023) and provides free, open-source, high quality standardised financial data to all participants. As discussed below in more detail, the data set is given in the form of pre-processed temporal tabular data and the task is the prediction of the relative performances of stocks within an evolving trading universe without access to the identity of individual stocks. Unlike other financial research papers that use proprietary data sets which can be difficult to validate [102, 101], this open financial data competition allows researchers to replicate findings transparently and allows us to focus on establishing ML end-to-end pipelines which achieve consistent profits trading a market-neutral portfolio under changing market regimes. Our pipeline, shown in Figure 3.1, is built upon simple, yet robust methodologies that avoid some of the problems of overfitting and high computational cost inherent to deep methods. To enhance the robustness of the pipeline, each step is implemented independently avoiding data leakage, so that the pre-processing and actual model do not share data. Key ingredients of our pipeline are the post-prediction processing, based on online learning techniques, which allow easy adaptation of models to regime changes without expensive retraining.

The chapter is organised as follows. Section 3.2 introduces the Numerai datasets used in this chapter. Section 3.3 describes and discusses the different computational methods, including online cross-validation, feature engineering and the different ML models considered and evaluated for the pipeline. Section 3.4 presents the results from our ML pipeline, including the impact of different design choices on the robustness of trading performance. Performances of ML models under different market regimes are discussed in Section 3.5. In Section 3.6, we introduce adaptations to our ML models based on online learning approaches, which help deal with regime changes, noting that these adaptations are generic and not limited to specific families of ML models. Lastly, we provide a summary and discussion, open directions and alternatives and provide a study of the robustness of our ML pipeline in Section 3.8.3.

Figure 3.1: **Schematic of the Machine Learning pipeline.** Starting with the Numerai data set, we consider feature engineering methods to augment the dataset and train an ML model (several are evaluated, including neural networks, but we settle for gradient boosting trees) to obtain the raw predictions. These then go through post-prediction processing (e.g., dynamic feature projection) to provide normalised predictions, which are then combined through model ensembling and dynamic model selection methods to output the predictions that are submitted to the Numerai tournament.

3.2 Numerai dataset and prediction task

Financial data are often available in the form of time series. These time series can be treated directly using classic methods such as ARIMA models [106] and more recently through deep learning methods such as Temporal Fusion Transformers [107]. However, such methods are easily overfitted and need expensive retraining for financial data, which are inherently affected by regime changes and high stochasticity. Alternatively, one can use feature engineering methods to transform these time series into *tabular form* through a process sometimes called 'de-trending' in the financial industry, where the characteristics of a financial asset at a particular time point, including features from its history, are represented by a data vector. In this representation, the time dimension is not considered explicitly, as the state of the system is captured through transformed features recomputed at each time point and the continuity of the temporal dimension is not used. For example, we can summarise the time series of the return of a stock with the mean and standard deviation over different look-back periods. Grouping these data rows for different financial assets into a table at a given time point we obtain a *tabular dataset*. If the features are informative, this representation can be used for prediction tasks at each time point, and allow us to employ robust and widely tested ML algorithms that are applicable to tabular data. The Numerai competition is based on a curated tabular data set with high-quality features made available to the research community.

Description of the dataset: The Numerai dataset is a temporal tabular dataset. A temporal tabular dataset is a collection of matrices $\{X_i\}_{1\leq i\leq T}$ collected over time eras 1 to T. Each matrix X_i represents data available at era i with shape $N_i \times M$, where N_i is the number of data samples in era i and M is the number of features describing the samples. The definition of the features is fixed throughout the eras, in the sense that the same computational formula is used to compute the features in each week, whereas the number of data samples N_i does not have to be constant across time. In the Numerai dataset considered here, the matrices X_i contain M stock market features (computed by Numerai) for N_i stocks, which are updated weekly (i.e., in our case the eras are weeks).

It is important to remark that the dataset is *obfuscated*, i.e., it is not possible for participants to know the identity of stocks or even which stocks are present each week. Each data row is indexed by a hash index, known only to Numerai, that maps the data rows to the stocks. As a result, it is not possible for participants to concatenate different data rows to create a continuous history of a stock. The matrix X_i thus provides a snapshot of the market at week i presented as an unknown set of stocks described by a

common set of features, such that the features are computed consistently across all stocks in the week and also computed consistently across different weeks.

The Numerai dataset starts on 2003-01-03 (Era 1). The tabular set has 1181 features, which are already normalised into 5 equal-sized integer bins, from 0 to 4. (Note that an initial release of the dataset had 1191 features but Numerai removed 10 ill-defined features in a subsequent refinement.) There are 28 target labels which are derived from stock returns using 14 proprietary normalisation methods (nomi, jerome, janet, ben, alan, paul, george, william, arthur, thomas, ralph, tyler, victor, waldo) each over 2 forward-looking periods (20 trading days, 60 trading days). The main target label to evaluate performance is target-nomi-v4-20, i.e., forward 20 trading days return obtained by the nomi normalisation method. This target label is scaled between 0 and 1, where a smaller value represents a lower forward return, and grouped into 5 bins (0, 0.25, 0.5, 0.75, 1.0) following a Gaussian-like distribution. For our analysis, we transform the features and labels so that both become zero-mean. For features, we subtract 2 from the integer bins so that the transformed bins are -2,-1,0,1,2. For the target labels, we subtract 0.5 so that the transformed targets are in the range -0.5 to 0.5).

Prediction task: The task in the Numerai tournament is to predict the *stock rankings* each week, ordered from lowest to highest expected return. The scoring is based on Spearman's rank correlation of the predicted rankings with the main target label (target-nomi-v4-20). Hence there is a single overall score each week regardless of the number of stocks to predict each week. Participants are not scored on the accuracy of the ranking of each stock individually, only on the overall score. Numerai uses the predicted rankings to construct a market-neutral portfolio which is traded every week (as of Dec 2022), i.e., the hedge fund buys and short-sells the same dollar amount of stocks. Therefore the relative return of stocks is more relevant than the absolute return, hence the prediction task is a ranking problem instead of a forecasting problem.

3.3 Methods

3.3.1 Robustness in Machine Learning pipelines

In this chapter, we aim to design an ML pipeline focusing on its robustness. Table 3.1 details issues related to robustness and reproducibility, as listed in a recent review [97], and how they are addressed here. By preventing look-ahead bias and other data leakage issues, our pipeline can be robustly applied to live trading setups.

In addition to avoiding data leakage, the following design choices are used to improve the robustness and reliability of the results. Firstly, the impact of random seeds is reduced by reporting results from average predictions over 10 different random seeds for each machine learning method. Secondly, the metrics used for model evaluation are the same as in the Numerai tournament to avoid researcher bias in discounting unfavourable results. Finally, cross-validation is independent of random seeds and any other human selection, thus reducing the chance of overfitting models to a particular data split.

For temporal datasets, standard cross-validation schemes cannot be used directly, as a random split of eras could lead to the training set including data that appears later in time than the validation and test sets, hence introducing look-ahead bias. To avoid this problem, we use *grouped time-series cross-validation*, which splits eras according to their chronological order (Fig. 3.2). Since for financial applications the target labels often involve future asset returns and are reported with a lag, we add a gap between the training and validation sets and similarly between the validation and test sets.

Issues affecting robustness of ML algorithms	*How the issue is addressed here*
'No test set'	A robust cross-validation scheme is used.
'Pre-processing on training and test set'	Numerai features are already standardised; hence minimal pre-processing.
'Feature selection on training and test set'	Feature Engineering is applied to each data row independently
'Duplicates in datasets'	A unique id for each data row reduces the chance of duplicates in dataset
'Model uses features that are not legitimate'	Only data provided by Numerai is used to train ML models—no extra features from other resources, and no cherry-picking of features.
'Temporal leakage'	We use *Grouped Time-Series Cross-Validation* with no overlap between training/validation/test (Fig. 3.2). Feature Engineering is applied to each data row independently, i.e., no data leakage between eras.
'Non-independence between training and test'	Training and test samples are market data at different periods without overlap.
'Sampling bias in test distribution'	The stocks trading each week are decided by Numerai based on operational and risk considerations.

Table 3.1: **Data analysis design.** Some common issues regarding data leakage in machine learning research [96, 97, 30] and how these issues are dealt with in this study.

Figure 3.2: Illustration of data split using grouped time-series cross-validation

3.3.2 Feature Engineering methods

We evaluate different feature engineering methods that can be applied to temporal tabular data sets with numerical features, as applicable to the Numerai data set. Firstly, new features can be created by applying polynomial transformations, such as multiplication and addition, to the original features. Here we create new features by multiplying two features, which can be thought of as capturing interactions between feature pairs. To alleviate the computational cost, when the number of features is large, we create new features from a random subset of feature pairs. Note that the computation of these features can be done in parallel for data from each era. Secondly, a simple way of data augmentation is to add randomness to the feature matrix with different dropout methods, which are used extensively to reduce overfitting in neural network models [23]. Here we apply dropout by multiplying the original data with a Boolean mask so that some numerical features are set to zero. The dropout is characterised by its sparsity level (how many features are set to zero) and its sparsity structure (how to choose the features set to zero). Since our tabular dataset has no local spatial structure, we use a random Boolean matrix with uniform probability. This encourages the ML methods to learn multiple feature relationships and reduces reliance on a small set of important features.

For our dataset, we first augment the feature matrix by creating additional features by multiplying feature pairs, followed by dropout with a random Boolean mask on the augmented feature matrix. A grid

search is used to find optimal hyperparameters of the feature engineering methods, i.e., the number of feature products and the sparsity of the dropout.

3.3.3 Machine Learning algorithms for tabular datasets

Numerous machine learning models have been proposed for tabular datasets, and different benchmarking studies have shown conflicting views on their performance [15, 23]. The biggest disagreement in the literature is whether gradient-boosting decision trees or neural networks are superior in regression and classification tasks of tabular datasets. Whereas one paper claims gradient boosting models (XGBoost) outperformed deep learning models in 8 out of 11 datasets and none of the deep learning models consistently outperform others [15], another paper suggests that well-tuned multi-layer perceptron (MLP) models with regularisation can outperform different gradient boosting models such as XGBoost and CatBoost [23]. Interestingly, both these studies share the view that neural networks with complicated designs, such as attention layers and other transformer layers, tend to generalise poorly, with a strong drop in performance when applied to data sets beyond their original study. Importantly, the Numerai data set is different from the data sets in the above benchmarking studies in that it is incremental instead of fixed. Hence the data distribution varies across time periods due to market regime effects, and we do not have a homogeneous distribution across cross-validation splits. With such a different problem setup, it is thus not possible to use the above benchmarking studies to guide our choice of ML method.

In this study, we benchmark a range of machine learning models, including different variants of gradient-boosting decision tree models and neural network models. The choice of ML models is based on the popularity of usage in data science competitions and code quality, as one of our aims is replicability of results. We train all machine learning models with a single GPU, the standard setup for most participants in data science competitions. Some brief details of the ML models used are provided in the following.

Gradient Boosting Decision Trees (GBDTs): Boosting can be seen as an extension of generalised additive models (GAM) where the additive components of smooth parametric functions can be replaced by any weak learners such as decision trees [108]. Historically, various boosting algorithms have been proposed for different loss functions. For example, AdaBoost [109] was proposed for binary classification problems with exponential loss, whereas Gradient Boosting was first proposed by Friedman in 2001 [110] for any smooth loss functions. Algorithm 10 in the SI outlines the iterative update equations of gradient boosting.

Of the various Python implementations of GBDTs, we use LightGBM [12] in this paper. CatBoost [13] is not used here as the Numerai dataset has no categorical features.

Algorithm 11 in the Supplementary Information (Section 3.8) shows how LightGBM implements gradient boosting with decision trees being the weak learners, including several computational and numerical improvements relative to XGBoost and other implementations. In addition to traditional gradient boosting decision trees (*LightGBM-gbdt*), we consider two other implementations of GBDT models:

- *Dropouts meet Multiple Additive Regression Trees (LightGBM-dart)* ignores a portion of trees when computing the gradient for subsequent trees [111], thus avoiding over-specialisation where the later learned trees can only affect a few data instances. This reduces the sensitivity of models towards decisions made by the first few trees.
- *Gradient-based One-Side Sampling (LightGBM-goss)* reduces the number of data instances used to build each tree: it keeps data instances with large absolute gradients and randomly samples a subset

of data with small absolute gradients. The approximation error of the gradient using LightGBM-goss converges to the standard method when the number of data is large, and it outperforms other data sampling (e.g., uniform sampling) in most cases.

For all LightGBM models, we use mean squared error (L2 loss) as the loss function for the regression problems. The number of trees and learning rate is optimised by hyperparameter searches. To prevent overfitting, the maximum depth and number of leaves in each tree and the minimal number of data samples in the leaves are tuned for each model. L1 and L2 regularisation are also applied. Data and feature sub-sampling are used to reduce similarities between trees: before building each tree, a random part of data is selected without re-sampling and a random subset of features is chosen to build the tree. For LightGBM-dart models, both the probability to apply dropout during the tree-building process and the portion of trees to be dropped out are tuned. Early stopping is applied using the validation dataset for LightGBM-gbdt models to further reduce overfitting.

Neural Networks (NNs): The most basic architecture of neural networks, multi-layer perceptron (MLP), has failed to outperform gradient boosting models in benchmark studies of tabular datasets [15]. Recently, more complex network architectures have been proposed for tabular data sets, as surveyed in [112] These new architectures can be classified into two major groups:

- *Hybrid models* that combine neural networks with other traditional ML methods, e.g., decision trees. Neural Oblivious Decision Ensembles (NODE) [113] is a generalisation of gradient boosting models into differentiable deep decision trees allowing end-to-end training with gradient descent optimisers such as PyTorch [114]. DeepGBM [115] combines two neural networks: CatNN to handle sparse categorical features, and GBDT2NN to distil tree structures from a pre-trained GBDT model to handle numerical features. A major limitation of these models is the large memory consumption, which makes them run out of memory on the NVIDIA 3080ti GPU. Therefore we do not use them in our benchmarking here.

- *Transformer-based models* that use deep attention mechanisms to model complex feature relationships. TabNet [19] uses sequential attention to perform instance-wise feature selection at each decision step, enabling interpretability and better learning. AutoInt [116] maps and models feature interactions in a low-dimensional space with a multi-head, self-attentive neural network with residual connections. AutoInt runs out of memory on a single GPU and is thus not used in our benchmarking. Tabnet also has similar memory issues, hence we down-sampled the data by keeping every fifth week of data (i.e., 20% of the original data) for the training/validation periods, so that Tabnet could be trained on the single GPU used in this study. Our aim is to compare performance under modest computational resources attainable by a wide class of users.

In summary, our benchmark analysis includes two NN models: MLP and TabNet implemented in PyTorch. We use mean squared error (L2 loss) for the regression problems.

3.4 Evaluation of Machine Learning methods for the Numerai temporal tabular data set

In this section, we study different ML methods applied to the Numerai temporal tabular data set for the prediction of stock rankings aimed at market-neutral stock portfolios.

Data Split We use the latest version (v4) of the Numerai dataset. The training period is fixed between 2003-01-03 (Era 1) to 2012-07-27 (Era 500); the validation period is fixed between 2012-12-21 (Era 521) and 2014-11-14 (Era 620); and the test period starts on 2015-05-15 (Era 646) and ends on 2022-09-23 (Era 1030). We apply a 1-year gap between training and validation periods to reduce the effect of recency bias so that the performance of the validation period will better reflect future performance. The gap between the validation period and test period is set to 26 weeks to allow for sufficient time to deploy trained machine learning models.

Evaluation of performance For each configuration of each ML method, we average over the predictions of different targets before scoring. The predictions are scored in each era by calculating the correlation (**Corr**) between the rank-normalised predictions and the actual (binned) stock ranking. The mean and standard deviation (volatility) of **Corr** are reported for both the validation and test periods. To measure the downside risk of the model, we also compute the *Maximum Drawdown*, defined as the largest drop suffered by an investor starting at any time during the validation/test period. As summary measures, we compute two standard ratios: (i) the Sharpe ratio, defined as the ratio of the mean and standard deviation of **Corr**; and (ii) the Calmar ratio, defined as the ratio of mean Corr over Maximum Drawdown. Good performance is characterised by large values of both of these ratios.

Model Training We use Optuna [117] to perform the hyperparameter search (see Section 3.8.2 in Supplementary Information) and select the hyperparameters with the highest Sharpe ratio for the main target (target-nomi-v4-20) in the validation period. The optimised hyperparameters for each ML method are then fixed, and we train 10 models, starting the algorithms from 10 different random seeds. We report the average prediction of these 10 models for evaluation.

Baseline Model As a baseline, we consider a factor momentum model created by linear combinations of signed features, where the sign of each feature is determined by the sign of the 52-week moving average of correlations of that feature with the target. This serves as a simple baseline *linear* model, which is then compared to the ML models that can capture non-linearity in the data.

Comparative results of the ML algorithms and Feature Engineering Table 3.2 shows the performance of the different ML algorithms with and without feature engineering on: (a) the validation set, and (b) the test set. We focus on methods that achieve the highest mean **Corr**, and Sharpe and Calmar ratios.

Firstly, almost all ML models performed substantially better than the factor momentum model (baseline), in both validation and test periods. Whereas the factor momentum model relies on linear relationships, the capability of ML models to learn non-linear relationships, in addition to linear ones, adds to their robustness and improved performance under different, often volatile, market regimes.

Secondly, we observe that Feature Engineering does not improve the performance of ML models. Although, in principle, Feature Engineering allows GBDT-based methods to model feature interactions more easily, our results suggest that these interactions are overfitted during the training process. For neural network-based models, feature engineering is not strictly necessary, as dropout is already embedded in network architectures.

Thirdly, we note that all ML models scored better in the validation period than the test period. This is expected, as it is well known that the performance of trading models deteriorates over time due to overcrowding and regime changes (a phenomenon known as alpha decay). Models that are overfitted to

(a) Performance over the validation period (2012-12-21 to 2014-11-14)

Method	Mean	Volatility	Max Draw	Sharpe	Calmar
Factor Momentum (baseline)	0.0229	0.0170	0.0691	1.3495	0.3314
MLP with FE	0.0423	0.0208	0.0241	2.0338	1.7552
MLP without FE	0.0443	0.0201	0.0065	2.2058	**6.8154**
TabNet without FE	0.0362	0.0189	0.0199	1.9125	1.8191
LightGBM-gbdt with FE	0.0483	0.0229	0.0307	2.1144	1.5733
LightGBM-gbdt without FE	0.0500	0.0224	0.0235	**2.2335**	2.1277
LightGBM-dart with FE	0.0496	0.0223	0.0215	**2.2274**	**2.3070**
LightGBM-dart without FE	0.0475	0.0199	0.0079	**2.3883**	**6.0127**
LightGBM-goss with FE	0.0288	0.0219	0.0687	1.3136	0.4192
LightGBM-goss without FE	0.0302	0.0234	0.0877	1.2877	0.3444

(b) Performance over the test period (2015-05-15 to 2022-09-23)

Method	Mean	Volatility	Max Draw	Sharpe	Calmar
Factor Momentum (baseline)	0.0080	0.0275	0.7877	0.2923	0.0102
MLP with FE	0.0237	0.0330	0.2912	0.7189	0.0814
MLP without FE	0.0258	0.0289	0.1668	**0.8931**	**0.1547**
TabNet without FE	0.0161	0.0296	0.5811	0.5431	0.0277
LightGBM-gbdt with FE	0.0253	0.0327	0.3064	0.7731	0.0826
LightGBM-gbdt without FE	0.0262	0.0321	0.2378	0.8140	0.1102
LightGBM-dart with FE	0.0265	0.0319	0.2151	**0.8313**	**0.1232**
LightGBM-dart without FE	0.0278	0.0284	0.1622	**0.9791**	**0.1714**
LightGBM-goss with FE	0.0169	0.0297	0.5539	0.5695	0.0305
LightGBM-goss without FE	0.0156	0.0318	0.7528	0.4896	0.0207

Table 3.2: Performance of different machine learning methods with and without feature engineering on the Numerai dataset for (a) validation period and (b) test period. The three top methods according to Sharpe ratio and Maximum Drawdown over the validation period are shown in italics in (a). The top method according to the Sharpe ratio and Maximum Drawdown over the test period is shown in boldface in (b). For TabNet, the pipeline with feature engineering cannot be run due to memory constraints.

recent training data will experience greater alpha decay than properly regularised models. To select the ML method, we consider the top models according to the Sharpe and Calmar ratios over the validation period: a high Sharpe ratio ensures the model has good overall performance, whereas a high Calmar ratio ensures good performance against the worst-case scenario, thus capturing the tail risks of the trading model. Indeed, we find that *LightGBM-dart without feature engineering* generalises well to the test period further into the future.

Finally, we note that LightGBM-gbdt has better generalisation to the test period than neural network-based models (TabNet), suggesting overfitting in these complex deep NN models. This indicates that although over-parameterised models can learn non-linear relationships in temporal tabular data sets, these relationships may be difficult to generalise under non-stationary data environments. Our results suggest that, despite their relative simplicity, gradient Boosting models capture non-linearity in a more robust and controlled manner, with early trees capturing linear relationships and non-linear relationships captured by the later trees, thus reducing the risk of catastrophic forgetting [118].

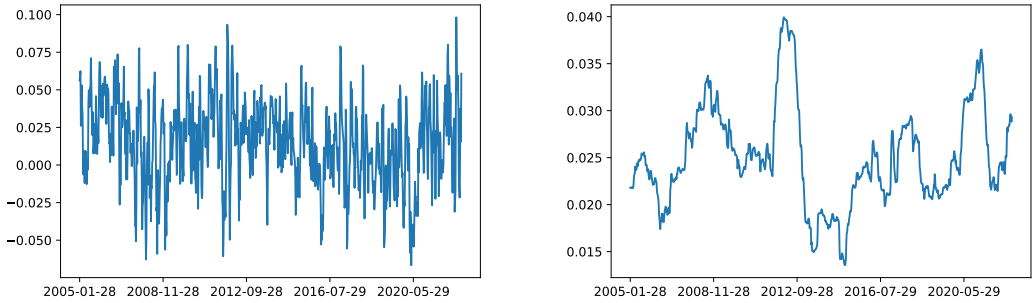
In summary, we find that the best performing model in our dataset is LightGBM-dart without feature

engineering. In the rest of the paper, we concentrate on this model to illustrate how the pipeline can be further improved with online learning to account for regime effects. However, to demonstrate the robustness of our pipeline and how it can be applied to improve the performance of any ML model, we will also report two other models: a similar GBDT model (LightGBM-gbdt without feature engineering) and a neural network model (MLP without feature engineering).

3.5 Dealing with regime effects in the ML pipeline

Financial data are heavily influenced by regime changes. Growth ('Bull') markets are characterised by low volatility and positive expected return, whereas high volatility and negative expected returns are characteristic of adverse ('bear') markets. Switches between regimes can be triggered by externalities, such as pandemics, economic recessions, etc. From the perspective of the Numerai data set, such regime effects affect model performance. Volatility is detrimental to long-term performance due to the negative compounding of investment losses, a phenomenon known as 'volatility tax'. Given that hedge funds are leveraged, we want to build consistent models with reasonably good performance under different market regimes, rather than models that have excellent performance in one market regime but fail in others.

We now focus on regime effects when using ML models for financial tabular temporal data sets. Specifically, we consider the effect of feature projection, and reducing the dependence on the initial trees in gradient boosting models.



(a) NMI

(b) NRVIX

Figure 3.3: **High and low volatility regimes in the Numerai data.** (a) Numerai Market Index (NMI) for the period between 2005-01-28 (Era 109) and 2022-09-23 (Era 1016); (b) the computed Numerai Realised Volatility Index (NRVIX) used to identify the high and low volatility regimes. The high volatility regime refers to weeks where NRVIX is higher than 0.025 and the low volatility regime refers to weeks where NRVIX is lower than 0.025.

Classification into high and low volatility regimes To classify the financial market into regimes, we consider an intrinsic measure derived directly from the Numerai dataset, as follows. We first compute the Numerai Market Index (NMI), i.e., the weekly performance of the baseline (linear) factor momentum portfolio, and we then calculate the Numerai Realised Volatility Index (NRVIX), defined as the standard deviation of NMI rolling over 52 weeks (Fig. 3.3). The eras are then classified into high and low volatility, based on a threshold of NRVIX=0.025, which is the mean over the first 7 years of data (2003-01-03 to 2010-02-26). According to this intrinsic characterisation, low volatility regimes have stable linear relationships of features to stock returns, often associated with a good performance by ML models. On the other hand, high Volatility regimes correspond to unstable linear relationships of features to stock returns, which lead to poor model performance. Figure 3.3 shows that the high and low NRVIX regimes are well aligned with macroeconomic events: high volatility regimes include the financial crisis (2007-2009), the Euro crisis (2011-2012), and the Covid pandemic (2020), whereas low volatility regimes correspond to benign market conditions with no significant macro event risks, during which the factor momentum baseline portfolio had good returns.

3.5.1 Feature Projection based on a fixed set of features

Feature projection is the elimination of the effect of particular features in the model, thus reducing the risk of over-relying on individual features. Because the predictive ability of individual features is highly dependent on market regimes, it is undesirable to have ML models with heavy (linear)-dependence on certain features since this can lead to long periods of drawdown if there is a regime change.

We start by evaluating the feature projection suggested by the Numerai tournament. Numerai recommends that participants reduce model dependencies on 420 'risky features' (out of the 1181 features). This risky features are projected out by subtracting the linear correlation, as follows. Given a week of data with n stocks, let $X \in \mathbb{R}^{n\times 420}$ be the matrix of risky features and $y \in \mathbb{R}^n$ the predicted rankings obtained from a model. For a given projection strength $\beta, 0 \leq \beta \leq 1$, the projected predicted ranking $\hat{y}$ is calculated as $\hat{y} = y - \beta XX^{\dagger}y$, where $X^{\dagger}$ is the pseudo-inverse of X. The Feature Neutral Correlation (FNC) is then calculated as the correlation of the projected predicted rankings. Using this procedure, we reduce the linear dependencies of predictions on the risky features.

(a) All regimes

Method	Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	Yes	0.0215	0.0182	0.1153	**1.1806**	**0.1865**
	No	0.0278	0.0284	0.1622	0.9791	0.1714
LightGBM-gbdt without FE	Yes	0.0204	0.0211	0.1998	0.9665	0.1021
	No	0.0262	0.0321	0.2378	0.8140	0.1102
MLP without FE	Yes	0.0179	0.0203	0.2606	0.8798	0.0687
	No	0.0258	0.0289	0.1668	0.8931	0.1547

(b) High Volatility

Method	Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	Yes	0.0227	0.0163	0.0223	**1.3888**	**1.0179**
	No	0.0314	0.0251	0.0657	1.2510	0.4779
LightGBM-gbdt without FE	Yes	0.0217	0.0198	0.0364	1.0953	0.5962
	No	0.0308	0.0293	0.1123	1.0497	0.2743
MLP without FE	Yes	0.0196	0.0193	0.0326	1.0191	0.6012
	No	0.0298	0.0276	0.1247	1.0802	0.2390

(c) Low Volatility

Method	Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	Yes	0.0206	0.0195	0.1153	**1.0576**	**0.1787**
	No	0.0252	0.0305	0.1622	0.8257	0.1554
LightGBM-gbdt without FE	Yes	0.0194	0.0220	0.1998	0.8820	0.0971
	No	0.0227	0.0338	0.2378	0.6727	0.0955
MLP without FE	Yes	0.0165	0.0210	0.2606	0.7875	0.0633
	No	0.0228	0.0296	0.1668	0.7721	0.1367

Table 3.3: **The effect of standard feature projection.** Performance of different ML methods on the Numerai dataset over the test period (2014-06-27 to 2022-09-23) with and without feature projection under different market regimes: the whole test period (All), high volatility regime (High Vol), and low volatility regime (Low Vol).

In Table 3.3, we compare the performance of the LightGBM-dart, LightGBM-gbdt and MLP with and without feature projection under different market regimes (all, high volatility, low volatility). The

projection strength β is set to 1 throughout. As expected, we find that the Volatility and Maximum Drawdown consistently reduced by feature projection across all models, suggesting an overall reduction of risk. Further, whereas feature projection improves the Sharpe and Calmar ratios of LightGBM-dart and LightGBM-gbdt under different market regimes, it does not improve the performance of MLP models consistently.

However, the feature projection procedure suggested by Numerai based on projection out a list of 'risky' features is not optimal. In Section 3.6, we will show how online learning techniques can be used to improve the procedure.

3.5.2 Pruning initial trees in Gradient Boosting models

For gradient-boosting tree models, there is a procedure related to feature projection, which can be used to reduce the excessive dependence on particular features. The procedure consists of pruning initial trees during prediction to reduce feature dependencies. Specifically, we perform a grid search over the number of initial trees to be pruned off in the trained LightGBM models, capping the number of pruned trees to not more than half of the trees to ensure our models do not degenerate.

(a) LightGBM-dart without FE

Pruned Trees	Mean	Volatility	Max Draw	Sharpe	Calmar
0	0.0278	0.0284	0.1622	0.9791	0.1714
100	0.0272	0.0264	0.1384	1.0293	0.1965
250	0.0264	0.0255	0.1299	1.0336	0.2032
500	0.0249	0.0238	0.1166	1.0459	0.2136

(b) LightGBM-gbdt without FE

Pruned Trees	Mean	Volatility	Max Draw	Sharpe	Calmar
0	0.0262	0.0321	0.2378	0.8140	0.1102
100	0.0265	0.0291	0.1835	0.9106	0.1444
250	0.0253	0.0259	0.1490	0.9769	0.1698
500	0.0253	0.0259	0.1490	0.9765	0.1698

Table 3.4: **The effect of tree pruning.** Performance of LightGBM models in the test period (2014-06-27 to 2022-09-23, i.e., All regimes) when pruning an increasing number of initial trees.

Table 3.4 compares the performance of the GBDT models (LightGBM-dart and LightGBM-gbdt) under pruning of different numbers of initial trees before feature projection. Pruning initial trees during prediction improves the Sharpe and Calmar ratios of both GBDT models, but LightGBM-gbdt models see a bigger improvement than LightGBM-dart models. This is expected as LightGBM-dart models already employ a similar idea during training, i.e., the trained trees in LightGBM-dart models are already optimised. Our numerical results also suggest that there is a natural upper limit to the trees to be pruned: there is little improvement in model performance if we prune above a threshold of 100-250 trees.

3.5.3 Joint effect of feature projection and tree pruning

We then considered the joint effect of feature projection and pruning initial trees. Table 3.5 compares the performance (FNC) of LightGBM-dart and LightGBM-gbdt models pruning a different number of initial trees *after feature projection*. The effect of pruning on model performance for both LightGBM models after feature projection is at best modest. As FNC is a measure of the effect of non-linear relationships, this suggests that in gradient boosting models, early weak learners (trees) mostly capture linear relationships whereas most of the non-linear relationships are captured in the later weak learners (trees). Therefore, pruning initial trees can be thought of as a model-dependent feature projection method.

(a) LightGBM-dart without FE with Feature Projection

Pruned Trees	Mean	Volatility	Max Draw	Sharpe	Calmar
0	0.0215	0.0182	0.1153	1.1806	0.1865
100	0.0208	0.0174	0.1079	1.1998	0.1928
250	0.0200	0.0168	0.1103	1.1918	0.1813
500	0.0183	0.0156	0.1044	1.1748	0.1753

(b) LightGBM-gbdt without FE with Feature Projection

Pruned Trees	Mean	Volatility	Max Draw	Sharpe	Calmar
0	0.0204	0.0211	0.1998	0.9665	0.1021
100	0.0206	0.0200	0.1912	1.0293	0.1077
250	0.0194	0.0188	0.2058	1.0307	0.0943
500	0.0193	0.0188	0.2063	1.0301	0.0936

Table 3.5: **The joint effect of feature projection and tree pruning.** Performance of the LightGBM models in the test period (2014-06-27 to 2022-09-23, All regimes) when pruning an increasing number of initial trees after feature projection.

3.6 Online Learning techniques to improve post-prediction processing

As a further improvement to the ML pipeline, we apply online learning approaches in two ways to deal with regime changes and non-stationarity. First, we introduce a version of feature projection called *dynamic feature projection*, which applies statistical rules to determine different subsets of features to project predictions in each era. Second, we introduce *dynamic model selection* to generate model ensembles by updating regularly the choice of model(s) from a model ensemble based on recent model performance.

The aim of online learning is to derive an optimal procedure to select ML models and parameters as data arrives continuously. In a continuous-time setting, the Hamilton-Jacobi-Bellman (HJB) equation is solved to find the optimal deterministic control for the decision problem [119]. The discrete-time equivalent, the Bellman equation, is used in reinforcement learning to derive optimal policies of agents [120].

For the Numerai tournament, we consider online learning in the discrete-time setting, since data and predictions are required once per week. For each week t ($1 \leq t \leq T$), we have a state (data) process X_t that contains all the information we know about the environment (Numerai datasets and trained

ML model parameters) up to week t. Our task is then to derive a deterministic decision process $D_t(\beta_t)$ described by parameters $\beta_t(X_t)$, subject to the objective function $V_T = \max_{D_t} \sum_{t=1}^{T} q(X_t, D_t)$, where $q(X_t, D_t)$ represents the utility at time t given the data and decision process.

Reinforcement Learning (RL) algorithms are commonly used to solve online learning problems. However, they are not used here due to the following reasons:

1. *Limited data:* Available data is not enough to train RL models, such as Deep Q Networks (DQN) [121], Proximal Policy Optimisation (PPO) [122] and Soft Actor-Critic (SAC) [123]). Generating a large number of samples is difficult here since we must avoid look-ahead bias.

2. *Expanding action space:* Most RL algorithms (see Ray-RLlib [103]) do not adapt naturally to an expanding action space, as is the case here. Indeed, for dynamic model selection, the number of potential models is unbounded, as newer models trained with the latest data can be added to the candidate list.

3. *Actions have negligible impact on environment:* Successful RL algorithms usually target applications (e.g., robotics and games [124]) where agent actions modify the environment, whereas for the trading models considered here, the trading activities have negligible market impact. RL algorithms thus reduce to an online learning prediction problem in our case.

4. *Large, correlated feature sets for projection:* The size of the set of risky features (420 features) and their high correlation makes it computationally infeasible to learn feature subsets through supervised ML or RL methods, as it is difficult to construct a robust reward function.

5. *Model ensembling can be simplified in our setting:* The model ensemble step of the pipeline assigns portfolio weightings to different ML models. Although similar to a multi-armed bandit problem, in our case no exploration is needed for the agent to learn the distribution of rewards from different choices since the performance of all ML models up to the decision time are known to the tournament participant.

As a consequence, instead of reinforcement learning algorithms, we use heuristics which are shown to be effective in improving the robustness of the ML pipeline. These heuristics can be interpreted as strong priors in Bayesian learning that greatly simplify our problem.

3.6.1 Dynamic Feature Projection

In Section 3.5.1, we presented the results of projecting out a subset of 'risky features', which was given by Numerai and fixed throughout the whole validation and test periods. However, since market conditions are variable, we suggest changing the set of features to project in each era to adapt our ML models without the need for expensive re-training of models. Specifically, each week we update the set of features to project based on rolling statistical properties of features, as follows. For each feature in the dataset, we calculate the correlation of the feature with the target (**Corr**) and compute lagged moving average statistics with a lag of 6 weeks to account for the lagged reporting of future performance. The look-back window to compute statistical properties of feature **Corr** is 52 weeks. We consider 5 different criteria to select the subset of features to be projected out:

1. 'Fixed': 420 features provided by the portfolio optimiser in Numerai (see Section 3.5.1).

2. 'Low Mean': 420 features that are least correlated to the target recently

3. ‘High Mean’: 420 features that are most correlated to the target recently

4. ‘Low Volatility’: 420 features that have correlations with lowest variability recently

5. ‘High Volatility’: 420 features that have correlations with largest variability recently

(a) LightGBM-dart without FE

Dynamic Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0215	0.0182	0.1153	1.1806	0.1865
Low Mean	0.0240	0.0164	0.0350	**1.4595**	**0.6857**
High Mean	0.0218	0.0185	0.0986	1.1783	0.2211
Low Vol	0.0244	0.0200	0.0538	1.2220	0.4535
High Vol	0.0226	0.0169	0.0341	1.3411	0.6628

(b) LightGBM-gbdt without FE

Dynamic Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0204	0.0211	0.1998	0.9665	0.1021
Low Mean	0.0234	0.0184	0.0495	1.2737	0.4727
High Mean	0.0199	0.0212	0.1469	0.9381	0.1355
Low Vol	0.0224	0.0228	0.1852	0.9797	0.1210
High Vol	0.0182	0.1633	0.0487	1.1986	0.4476

(c) MLP without FE

Dynamic Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0179	0.0203	0.2606	0.8798	0.0687
Low Mean	0.0211	0.0185	0.0806	1.1387	0.2618
High Mean	0.0186	0.0201	0.1283	0.9256	0.1450
Low Vol	0.0206	0.0215	0.0878	0.9598	0.2346
High Vol	0.0191	0.0172	0.0730	1.1150	0.2616

Table 3.6: **The effect of Dynamic Feature Projection.** Performance of different ML models in the test period (2014-06-27 to 2022-09-23) with different dynamic feature projection methods. Fixed corresponds to the standard Feature Projection in Section 3.5.1.

Table 3.6 compares the performance of LightGBM-dart, LightGBM-gbdt and MLP models under each of the five dynamic feature projection schemes. All Dynamic Feature Projection methods perform better than using a fixed set of features but the ‘Low Mean’ projection method has the best Sharpe and Calmar ratios for all ML models, followed by projection of ‘High Volatility’ features. The worse performance of ‘High Mean’ and ’Low Volatility’ projections suggests that a large part of the model risks can be attributed to recently under-performing and high volatility features.

Next we compared the performance of the different dynamic feature projection schemes under low volatility and high volatility market regimes, as defined in Section 3.5. The results can be found in Tables 3.9 and 3.8 in the Supplementary Information. Whereas projecting out ‘Low Mean’ features performs best in low volatility regimes, projecting out ‘Low Volatility’ and ’High Mean’ features performs best in high volatility regimes. This suggests that in a low volatility regime, factors that are performing well recently continue to do so in the near future as the feature correlation structure is more stable. On the other hand, in a high volatility regime, the ML models after projecting out of ‘Low Volatility’ and ’High Mean’

features lead to improved performance. 'Low Volatility' represents features that have a low variance and stable performance in the last 52 weeks. During volatile regimes, these features will under-perform; hence models that project out these features have improved performance when there is market stress.

3.6.2 Dynamic Model Selection

In practice, it is not possible to know the best dynamic feature projection in advance. Therefore, we propose an online learning procedure consisting of two steps to select the dynamic feature projection. First, we have a warm-up period to collect data on model performances, during which all 5 feature projection methods (fixed, low mean, high mean, low volatility, high volatility) have equal weighting. The second step is to allocate weights to the optimal model based on recent performance according to the following criteria:

- 'Average': Using all five feature projection methods with equal weighting
- 'Momentum': Using the feature projection method with the highest Mean **Corr** in the last 52 weeks
- 'Sharpe': Using the feature projection method with the highest Sharpe Ratio in the last 52 weeks
- 'Calmar': Using the feature projection method with the highest Calmar ratio in the last 52 weeks

(a) LightGBM-dart without FE

Model Selection	Mean	Volatility	Max Draw	Sharpe	Calmar
Average	0.0229	**0.0160**	0.0619	**1.4323**	0.3700
Momentum	**0.0246**	0.0180	0.0533	1.3654	0.4615
Sharpe	0.0234	0.0165	0.0533	1.4148	0.4390
Calmar	0.0225	0.0171	**0.0350**	1.3122	**0.6429**

(b) LightGBM-gbdt without FE

Model Selection	Mean	Volatility	Max Draw	Sharpe	Calmar
Average	0.0216	0.0177	0.0710	1.2165	0.3042
Momentum	0.0228	0.0201	0.0729	1.1342	0.3128
Sharpe	0.0224	0.0187	0.0729	1.1966	0.3073
Calmar	0.0216	0.0195	0.0508	1.1102	0.4252

(c) MLP without FE

Model Selection	Mean	Volatility	Max Draw	Sharpe	Calmar
Average	0.0195	0.0175	0.0918	1.1149	0.2124
Momentum	0.0212	0.0191	0.0878	1.1124	0.2415
Sharpe	0.0207	0.0186	0.0878	1.1110	0.2358
Calmar	0.0187	0.0201	0.1973	0.9309	0.0948

Table 3.7: **The effect of dynamic model selection.** Performance of different ML models in the test period (2014-06-27 to 2022-09-23) with different online learning procedures selecting the optimal dynamic feature projection method.

In Table 3.7, we use these criteria to select the optimal dynamic feature projection based on recent performance. As above, a lag of 6 weeks is applied to account for data delays. We find that for all three ML models (LightGBM-dart/LightGBM-gbdt/MLP), the 'Momentum' selection method has higher 'Mean' Corr and Calmar ratio than the'Average' and 'Sharpe' methods. This shows that the 'Momentum' method, a very simple model selection method that chooses the recent best-performing model, can adapt a trained ML model towards different market regimes efficiently. However, we find that the simple 'Average' scheme has the lowest 'Volatility' and highest Sharpe ratio. As expected, the 'Calmar' selection method gives a lower Maximum Drawdown and higher Calmar ratio than all the other dynamical model selection algorithms. However, for MLP models, the 'Calmar' selection method significantly underperforms, with a much higher Max Drawdown. This suggests that selection based on historical drawdown is not robust, especially under situations with regime changes.

In summary, the proposed online learning procedure to select optimal dynamic feature engineering methods can significantly reduce trading risks and improve the robustness of trading models by considering simple schemes to select or average the different dynamic feature projection methods.

3.7 Discussion

Summary: Motivated by the Numerai tournament, we have designed an ML pipeline that can be applied to tabular temporal data of stock prices to underpin strategies for trading of market-neutral stock portfolios. The various steps in the ML pipeline are designed for robustness against regime changes and to avoid information leakage through time by using models with relatively low complexity, so as to reduce the danger of overfitting, and with high robustness to changes in hyperparameters and other choices in the algorithms.

Regarding the choice of ML models, we find that gradient-boosting decision tree models are both more robust and interpretable than neural network-based models, and they allow more consistent performance under different market regimes.

We also find that post-prediction processing using online learning techniques, which are model-agnostic, is an effective means of adapting trained ML models towards new situations without the need to retrain ML models and introduce additional model uncertainty. Firstly, using dynamic feature projection produces models with different flavours in an interpretable way, which also have better risk-adjusted performance than models with fixed feature projection. Secondly, dynamic model selection can be applied in guiding the selection of an optimal model(s) from a growing model ensemble. We find that a simple design, such as equal-weighted models, has a robust performance under different market regimes, but selecting the best model based on recent performance can provided an improvement as it switches to a lower-risk model during more volatile market regimes.

Robustness of the pipeline: An important consideration in this work is to provide a methodology that is robust to diverse sources of variability. Although not presented in the main text, in Section 3.8.3 of the Supplementary Information, we have carried out an extensive study of the robustness of our ML pipeline under three different sources of variability: (i) different random seeds in the training of algorithms; (ii) changes in data splits for cross-validation; and (iii) randomness in feature selection for feature projection. The results show that LightGBM dart models are robust against these changes. The statistical rules used in dynamic feature projection are also shown to perform better than features chosen at random.

Further work: Finally, we discuss some ideas for further work to improve the ML pipeline presented here.

Staking is commonly used in ML competitions to improve the robustness of models. The dynamic model selection method suggested in this study falls into this broad category, whereby different models can be chosen based on certain statistical criteria that mirror different strategies under regime changes. It remains an open research area into how reinforcement learning or other online learning methods can be used to learn optimal staking weights between different ML models, given their historical performance and correlations.

The diversity of models within a model ensemble is a key ingredient for dynamic model selection and other model ensemble/staking methods. A new metric could be designed to study the impact of a new ML model on an existing model ensemble. This metric could then be used to train new ML models that are uncorrelated to existing ones.

We also note that although simple feature engineering methods did not improve performance here (see Table 3.2), an open direction could be to identify robust relationships between features over different regimes using generative models (e.g., Variational Autoencoders [125]) to create new features that could capture meaningful and non-trivial non-linear relationships.

Conclusion: Overall, our results suggest using simple, well-established ML models, such as gradient-boosting decision trees, instead of specialised neural network models for this task. Rather than using a single neural network to perform feature engineering, model training/inference and post-prediction transformations, our modularised design of the ML pipeline offers increased robustness and transparency. Researchers can add, modify or delete a component without affecting the rest of the pipeline. Creating model ensembles improves model performance by reducing idiosyncratic variance from individual ML models. The simple model selection rules based on recent performance provide a baseline that works well under different market regimes, whereas various portfolio metrics such as Sharpe and Calmar ratios are improved by using the recently best-performing models.

The Gradient Boosting models used here are suitable for distributed learning, where large datasets are split into smaller batches to train on different machines with varying computational limitations. Data science competitions like the Numerai tournament rely on community efforts of individual data scientists to create a meta-model. Such crowdsourcing relies on the assumption that a complicated ML model, which would need to be trained with advanced hardware, can be approximated by combining a number of simpler ML models (each trained with fewer data or features). Studying the convergence of model performance would be important for organising the data science competition as it decides how many participants are needed to maintain a well-diversified pool of models to create the meta-model.

3.8 Supplementary Information

3.8.1 Dynamic Feature Projection for low and high volatility regimes

Low Volatility

(a) LightGBM-dart without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0206	0.0195	0.1153	1.0576	0.1787
Low Mean	0.0255	0.0175	0.0350	**1.4578**	**0.7286**
High Mean	0.0207	0.0206	0.0986	1.0033	0.2099
Low Vol	0.0238	0.0221	0.0538	1.0793	0.4424
High Vol	0.0235	0.0180	0.0341	1.3069	0.6891

(b) LightGBM-gbdt without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0194	0.0220	0.1998	0.8820	0.0971
Low Mean	0.0251	0.0188	0.0495	1.3328	0.5071
High Mean	0.0184	0.0228	0.1469	0.8053	0.1253
Low Vol	0.0214	0.0247	0.1852	0.8657	0.1150
High Vol	0.0225	0.0188	0.0487	1.1939	0.4620

(c) MLP without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0165	0.0210	0.2606	0.7875	0.0633
Low Mean	0.0215	0.0187	0.0496	1.1496	0.4335
High Mean	0.0170	0.0210	0.1283	0.8118	0.1325
Low Vol	0.0194	0.0229	0.0878	0.8487	0.2210
High Vol	0.0194	0.0177	0.0730	1.0990	0.2658

Table 3.8: Performance of ML models in the test period (2014-06-27 to 2022-09-23) with different dynamic feature projection methods in low volatility regime

High Volatility

(a) LightGBM-dart without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0227	0.0163	0.0223	1.3888	1.0179
Low Mean	0.0220	0.0148	0.0199	1.4907	1.1055
High Mean	0.0233	0.0151	0.0206	**1.5372**	**1.1311**
Low Vol	0.0252	0.0168	0.0330	1.4980	0.7636
High Vol	0.0215	0.0152	0.0143	1.4077	1.5035

(b) LightGBM-gbdt without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0217	0.0198	0.0364	1.0953	0.5962
Low Mean	0.0212	0.0176	0.0380	1.2039	0.5579
High Mean	0.0218	0.0186	0.0334	1.1728	0.6527
Low Vol	0.0237	0.0201	0.0306	1.1792	0.7745
High Vol	0.0209	0.0173	0.0308	1.2068	0.6786

(c) MLP without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Fixed	0.0196	0.0193	0.0326	1.0191	0.6012
Low Mean	0.0205	0.0183	0.0806	1.1212	0.2543
High Mean	0.0170	0.0210	0.1283	0.8118	0.1325
Low Vol	0.0222	0.0194	0.0397	1.1442	0.5592
High Vol	0.0187	0.0165	0.0336	1.1368	0.5565

Table 3.9: Performance of ML models in the test period (2014-06-27 to 2022-09-23) with different dynamic feature projection methods in high volatility regime

3.8.2 Hyperparameter search space for different ML models

We ran all experiments on a GPU cluster, each node of which contains a NVIDIA GeForce RTX 2080 Ti GPU, running with 4352 CUDA cores and 11GB memory. Hyperparameter search is performed using Optuna [117]. For each Feature Engineering/ML pipeline, hyperparameter search is ran for at most 8 hours or at most 100 configurations, whichever came first. The default TPE sampler in Optuna is used to perform the hyperparameter search.

In Figure 3.4 and 3.5, we list the hyperparameter search parameters defined in Optuna [117] for different ML models used in the main text to train the models. For MLP and TabNet, we use Adam [126] as the gradient optimiser, using the recommended learning rates in their implementations.

- Feature Engineering
  - Numerai Basic Feature Engineering
    - dropout pct: low:0.05, high:0.25, step:0.05,
    - no product features: low:50, high:1000, step:50,
- ML Models
  - LightGBM-gbdt
    - n estimators: low:50, high:1000, step:50
    - learning rate: low:0.005, high:0.1, log:True
    - min data in leaf: low:2500, high:40000, step:2500
    - lambda l1: low:0.01, high: 1, log:True
    - lambda l2: low:0.01, high: 1, log:True
    - feature fraction: low:0.1, high:1, step:0.05
    - bagging fraction: low:0.5, high:1, step:0.05
    - bagging freq: low:10, high:50, step:10
  - LightGBM-dart
    - n estimators: low:50, high:1000, step:50
    - learning rate: low:0.005, high:0.1, log:True
    - min data in leaf: low:2500, high:40000, step:2500
    - lambda l1: low:0.01, high: 1, log:True
    - lambda l2: low:0.01, high: 1, log:True
    - feature fraction: low:0.1, high:1, step:0.05
    - bagging fraction: low:0.5, high:1, step:0.05
    - bagging freq: low:10, high:50, step:10
    - drop rate: low:0.1, high:0.5, step:0.1
    - skip drop: low:0.1, high:0.8, step:0.1
  - LightGBM-goss
    - n estimators: low:50, high:1000, step:50
    - learning rate: low:0.005, high:0.1, log:True
    - min data in leaf: low:2500, high:40000, step:2500
    - lambda l1: low:0.01, high: 1, log:True
    - lambda l2: low:0.01, high: 1, log:True
    - feature fraction: low:0.1, high:1, step:0.05
    - bagging fraction: low:0.5, high:1, step:0.05
    - bagging freq: low:10, high:50, step:10
    - top rate: low:0.1, high:0.4, step:0.05
    - other rate: low:0.05, high:0.2, step:0.05

Figure 3.4: Hyperparameter Space for ML models

- Machine Learning
  - MLP
    - max epochs: low:10, high:100, step:5
    - patience: low:5, high:20, step:5
    - num layers: low:2, high:7, step:1
    - neurons: low:64, high:1024, step:64
    - neuron scale: low:0.3, high:1, log:True
    - dropout: low:0.1, high:0.9, log:True
    - batch size: low:10240, high:40960, step:10240
  - TabNet
    - max epochs: low:10, high:100, step:5
    - patience: low:5, high:20, step:5
    - batch size: low:1024, high:4096, step:1024
    - num d: low:4, high:16, step:4
    - num a: low:4, high:16, step:4
    - num steps: low:1, high:3, step:1
    - num shared: low:1, high:3, step:1
    - num independent: low:1, high:3, step:1
    - gamma : low:1, high:2, step:0.1
    - momentum: low:0.01, high:0.4, step:0.01
    - lambda sparse: low:0.0001, high:0.01, log:True

Figure 3.5: Hyperparameter Space for ML models

3.8.3 Robustness of ML pipeline

One of the aims in this work was to provide a robust pipeline for tabular temporal data under regime changes. Here we present additional results of the robustness of the method under different scenarios and sources of variability.

Robustness under changes of random seeds in the learning algorithms In Table 3.10, we report the variability of the performance of the LightGBM-dart, LightGBM-gbdt and MLP models trained starting from 10 different initial random seeds. The performance is generally robust to the change in random seeds, with small variances in the prediction of the mean **Corr** and volatility and moderate for the Maximum Drawdown.

Model		Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	Mean	0.0254	0.0266	0.1567	0.9593	0.1639
	S.D.	0.0006	0.0007	0.0158	0.0365	0.0175
LightGBM-gbdt without FE	Mean	0.0253	0.0312	0.2338	0.8104	0.1100
	S.D.	0.0006	0.0006	0.0296	0.0278	0.0153
MLP without FE	Mean	0.0233	0.0271	0.1643	0.8600	0.1446
	S.D.	0.0009	0.0011	0.0248	0.0365	0.0219

Table 3.10: Variability of the performance of ML models in the test period (2015-05-15 to 2022-09-23). The mean and standard deviation of each portfolio metrics are calculated over models with 10 different random seeds for each method

A general strategy to reduce the variance of the predictions is to combine different ML models. There are two ways to do so: (i) averaging over models, i.e., averaging the performance of different models; and (ii) averaging over predictions, i.e., averaging predictions from each model and then scoring the average prediction from different models against the target. In Table 3.11, we apply these two types of averaging to the 10 models obtained training with different initial random seeds, whose average and variance predictions are reported in Table 3.10 To do so, we create 6 subsets of 5 models, e.g., subset 1 contains the models with random seeds 1-5; subset 2 contains the models with random seeds 2-6; etc; and subset 6 contains the models with random seeds 6-10. We find that averaging over predictions from the different models gives higher mean **Corr** and Sharpe/Calmar ratios. Therefore, this averaging method is used to compute model performances in Table 3.2 in the main text. Model variances of the averaged predictions are also much lower than the individual models.

Model	Average		Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	Over Models	Mean	0.0255	0.0259	0.1500	0.9832	0.1704
		S.D.	0.0002	0.0001	0.0081	0.0094	0.0102
	Over Predictions	Mean	0.0276	0.0282	0.1605	0.9765	0.1721
		S.D.	0.0002	0.0001	0.0087	0.0089	0.0104
LightGBM-gbdt without FE	Over Models	Mean	0.0253	0.0308	0.2233	0.8196	0.1134
		S.D.	0.0001	0.0002	0.0080	0.0062	0.0044
	Over Predictions	Mean	0.0260	0.0319	0.2306	0.8161	0.1131
		S.D.	0.0001	0.0002	0.0083	0.0061	0.0044
MLP without FE	Over Models	Mean	0.0231	0.0259	0.1490	0.8922	0.1550
		S.D.	0.0003	0.0003	0.0055	0.0095	0.0065
	Over Predictions	Mean	0.0254	0.0284	0.1610	0.8930	0.1576
		S.D.	0.0002	0.0004	0.0062	0.0078	0.0050

Table 3.11: Performance of different ML methods on Numerai v4 dataset in the test period (2015-05-15 to 2022-09-23) with different averaging methods

Robustness under different cross-validation data splits As financial data are regime dependent, an important measure of model robustness is to measure the performance of ML models that have been trained using different cross-validation splits of the data and compute how much the model performance changes over different test periods.

To ascertain the robustness of data splits, we have carried out 3 cross-validation splits (CV 1, CV 2, CV 3) as shown in Table 3.12. The hyperparameters are optimised under CV 1, which is the cross-validation used to generate the model performances in the main text. These hyperparameters are fixed for the models trained under the CV 2 and CV 3 splits. For ML methods that require early stopping, the data in the validation period (different for each split) are used to regularise the models. Therefore, by reusing the optimised hyperparameters across all splits, we evaluate the robustness of the model performance to the optimisation of hyperparameters. We then compute the performance when applying the models to shifted cross-validation datasets in the walk-forward CV 2 and CV 3 data splits. Our results show good consistency in performance across CV 2 and CV 3, with only a small deterioration of the results as compared to CV 1 (over which the hyperparameters were optimised). We also find that LightGBM-dart with FE, the ML method that has the highest mean **Corr** in CV 1, has the greatest return and best Sharpe and Calmar ratios also in other cross-validations, as seen in Table 3.13.

	Train Start	Train End	Validation Start	Validation End	Enter Ensemble
CV 1	2003-01-03	2012-07-27	2012-12-21	2014-11-14	2015-05-15
CV 2	2003-01-03	2014-06-27	2014-11-21	2016-10-14	2017-04-14
CV 3	2003-01-03	2016-05-27	2016-10-21	2018-09-14	2019-03-15

Table 3.12: Various cross-validation schemes to train ML models on different parts of the data. CV 1 is the cross-validation used for hyperparameter optimisation and training ML models in the main text.

(a) CV 1 (2015-05-15 to 2022-09-23)

Method	Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	0.0278	0.0284	0.1622	0.9791	0.1714
LightGBM-gbdt without FE	0.0262	0.0321	0.2378	0.8140	0.1102
MLP without FE	0.0258	0.0289	0.1668	0.8931	0.1547

(b) CV 2 (2017-04-14 to 2022-09-23)

Method	Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	0.0250	0.0278	0.1817	0.8990	0.1376
LightGBM-gbdt without FE	0.0231	0.0324	0.3227	0.7104	0.0716
MLP without FE	0.0215	0.0289	0.2307	0.7446	0.0932

(c) CV 3 (2019-03-15 to 2022-09-23)

Method	Mean	Volatility	Max Draw	Sharpe	Calmar
LightGBM-dart without FE	0.0264	0.0297	0.1380	0.8140	0.1913
LightGBM-gbdt without FE	0.0261	0.0336	0.1584	0.7772	0.1648
MLP without FE	0.0224	0.0240	0.1171	0.9339	0.1913

Table 3.13: Performance of selected machine learning methods on the Numerai dataset in the test period for various walk-forward cross-validation schemes, (a) CV 1, (b) CV 2 and (c) CV 3

Robustness under feature selection for dynamic feature projection A fixed set of 420 features to be projected was given by the Numerai organisers based on internal evaluations of parameters. In Section 3.6, we introduce several statistical rules that allow us to select a varying subset of features to be projected in each era based on empirical heuristic criteria motivated by financial modelling.

To evaluate the robustness of the proposed statistical rules, we draw 100 subsets of 420 features selected at random. and use each set to project the raw predictions from ML models. We then evaluate the performance of ML models based on each of the random subsets. Using the procedure described in section 3.6.2 we then select the optimal dynamic feature projection method and compute the performance of the top 10 models of the highest mean **Corr**, Sharpe and Calmar ratio over the test period. The results are reported in Table 3.14 and should be compared to the performance of the same models in Table 3.7, which were obtained with dynamic feature projection using the statistical rules defined in section 3.6.2.

The mean **Corr** of models obtained with random feature projection for each rule (Momentum/Sharpe/-Calmar) are lower than those obtained using the statistical rules in Table 3.7. On the other hand, the Sharpe ratio of models for models with random feature projection is slightly higher, as expected due to the variance reduction effect by averaging over 10 different models. For models selected based on the Calmar rule, the models obtained with statistical rules have a much higher Calmar ratio than random feature projection. It suggests the statistical rules defined can effectively reduce model risks by reducing linear exposure to undesirable features.

(a) LightGBM-dart without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Average	0.0214	0.0147	0.0482	1.4547	0.4440
Momentum	0.0216	0.0149	0.0472	1.4522	0.4576
Sharpe	0.0213	0.0147	0.0459	1.4474	0.4641
Calmar	0.0214	0.0148	0.0453	1.4504	0.4724

(b) LightGBM-gbdt without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Average	0.0203	0.0167	0.0664	1.2140	0.3057
Momentum	0.0208	0.0167	0.0641	1.2457	0.3245
Sharpe	0.0206	0.0168	0.0618	1.2267	0.3333
Calmar	0.0216	0.0195	0.0508	1.1102	0.2743

(c) MLP without FE

Feature Projection	Mean	Volatility	Max Draw	Sharpe	Calmar
Average	0.0176	0.0165	0.0831	1.0658	0.2118
Momentum	0.0179	0.0165	0.0790	1.0842	0.2266
Sharpe	0.0177	0.0164	0.0762	1.0751	0.2323
Calmar	0.0175	0.0167	0.0825	1.0511	0.2121

Table 3.14: Performance of different ML models in the test period (2015-05-15 to 2022-09-23) obtained with random feature projection. These are averages obtained by selecting the top 10 models under the different online learning procedures over the test period.

3.8.4 Pseudocode for algorithms used in the chapter

For completeness, we present here brief pseudocode for some of the main methods in the chapter with the appropriate references.

Algorithm 10: Gradient boosting algorithm [110, 127]

Given N data samples $(\mathbf{x_i}, y_i), 1 \leq i \leq N$ with the aim to find an increasing better estimate $\hat{f}(\mathbf{x})$ of the minimising function $f(x)$ which minimise the loss $\mathcal{L}(f)$ between targets and predicted values. $\mathcal{L}(f) = \sum_i l(y_i, f(\mathbf{x_i}))$ where l is a given loss function such as mean square losses for regression problems. Function f is restricted to the class of additive models $f(\mathbf{x}) = \sum_{k=1}^{K} w_k h(\mathbf{x}, \boldsymbol{\alpha_k})$ where $h(\cdot, \boldsymbol{\alpha})$ is a weak learner with parameters $\boldsymbol{\alpha}$ and w_k are the weights.

Initialise $f_0(\mathbf{x}) = \arg\min_{\boldsymbol{\alpha_0}} \sum_{i=1}^{N} l(y_i, h(\mathbf{x_i}, \boldsymbol{\alpha_0}))$

for *k = 1 : K* **do**

- Compute the gradient residual using $g_{ik} = -\left[\frac{\partial l(y_i, f_{k-1}(\mathbf{x_i}))}{\partial f_{k-1}(\mathbf{x_i})}\right]$
- Use the weak learner to compute $\boldsymbol{\alpha_k}$ which minimises $\sum_{i=1}^{N}(g_{ik} - h(\mathbf{x_i}, \boldsymbol{\alpha_k}))^2$
- Update with learning rate λ $f_k(\mathbf{x}) = f_{k-1}(\mathbf{x}) + \lambda h(\mathbf{x}, \boldsymbol{\alpha_k})$

end

Return $f(\mathbf{x}) = f_K(\mathbf{x})$

Algorithm 11: Gradient boosting tree algorithm implemented in LightGBM [128, 110, 127]

Initialise $f_0(\mathbf{x}) = \arg\min_{\boldsymbol{\alpha_0}} \sum_{i=1}^{N} l(y_i, \mathbf{x}, \boldsymbol{\alpha_0})$

for *k = 1 : K* **do**

- For $i = 1, 2, \ldots N$, compute the gradient residual using $g_{ik} = -\left[\frac{\partial l(y_i, f_{k-1}(\mathbf{x_i}))}{\partial f_{k-1}(\mathbf{x_i})}\right]$
- Fit a decision tree to the targets g_{ik} giving terminal leaves R_{kj}, $j = 1, 2, \ldots J_k$, where J_k is the number of terminal leaves.
- For $j = 1, 2, \ldots J_k$, compute $\alpha_{jk} = \arg\min_{\boldsymbol{\alpha}} \sum_{\mathbf{x_i} \in R_{kj}} l(y_i, f_{k-1}(\mathbf{x_i}) + \boldsymbol{\alpha})$
- Update boosting trees with learning rate λ $f_k(\mathbf{x}) = f_{k-1}(\mathbf{x}) + \lambda \sum_{j=1}^{J_k} \alpha_{kj} \mathbb{I}(\mathbf{x} \in R_{kj})$

end

Return $f_K(x)$

Chapter 4

Deep incremental learning models for financial temporal tabular datasets with distribution shifts

4.1 Introduction

Many important applications of machine learning (ML), such as the Internet of Things (IoT) [129] and cyber-security [130], 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 [1].

Incremental learning (IL) techniques [131, 132, 133] are used to adapt deployed machine learning systems to changes in data streams. For example, in image classification systems, class incremental learning [134, 135, 136] 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) [137] which results in model degradation [138] 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) [139, 140, 141] 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 [142]. 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 [143], self-driving cars [144], or robotics [145]) remains a challenge, and complex algorithms have been introduced to bridge the gap between controlled environments and real-life situations [146, 147]. Hence the applicability of RL models can suffer from lack of robustness[148] and interpretability [149] 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 [150] 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 [151] 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 [152]. 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. 4.1.

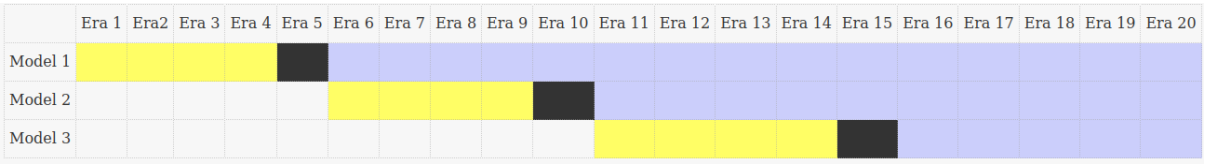


Figure 4.1: Schematic of how model predictions are reused in an incremental learning model. Consider three models (Models 1-3) each trained over a training period of 4 eras (weeks) and with a lag of 1 era (week). Model 1 is trained using information (both features and targets) up to Week 4 and after 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 [1, 137] 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 [153]. For instance, Deep Regression Ensemble (DRE) [154] 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.

4.2 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.

4.2.1 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.

4.2.2 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

$$y_t = (1 - \alpha)y_{t-1} + \alpha x_t \tag{4.1}$$

with initialisation $y_1 = x_1$.

Remark. More complex time series models have been developed, including sequence models in deep learning, such as LSTM [155] and Transformers [107]. However, these models tend to be overparameterised and lack robustness to regime changes [156]. They also involve heavy computational costs associated with the training and updating of models.

4.2.3 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 [157, 158, 159] are deterministic transformations, recently proposed by Lyons, which can extract features at increasing orders of complexity from multivariate data, including time series. See [159] 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 4.9.1 in the Supplementary Information.
- *Random Fourier Transform (RFT)*: RFTs have been used in [160] 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 [161]. Details on how RFTs are applied on the Numerai dataset are given in the Supplementary Information, see Algorithm 23 in Section 4.9.1.

Remark. As discussed in Section 4.3.2 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 [151], without relying on complex and expensive advanced neural network architectures such as Recurrent Neural Networks (RNN), Long-Short-Term-Memory (LSTM) Networks or Transformers [107].

4.3 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.

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

Factor-timing models [162] 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 (4.1)) are converted into normalised rankings, which are then used as weights for the linear factor-timing model (see Algorithm 12). 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 12 avoiding look-ahead bias.

Algorithm 12: 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 $\hat{z}_t = X_t r_t$

4.3.2 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 [16, 15, 14, 23]. Several benchmarking studies [16, 15, 14] have demonstrated that advanced deep learning methods, such as transformers [19] 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 [16] 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 [16] 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 [151, 15, 14].

In this paper, GBDT models are studied in detail for tabular prediction, as it has demonstrated strong performances in benchmarking studies [16, 15, 14, 151] 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.

4.4 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. 4.1).

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 13 outlines the overall structure of the computational framework.

Algorithm 13: Deep IL model with model stacking

Input: Temporal Tabular Dataset $\{X_i, y_i\}_{1 \le i \le T}$, 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 $1 \le l \le L$ **do**

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

  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 $1 \le k \le K_L$ **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 [163], and Deep Learning models for sequences, such as LSTM [164], 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 [165], hyperparameters [166] and architectures [167] 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.

4.5 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 [106, 107], 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) [27, 28] 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 4.2, 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. [151].

[1] The data keeps updating every week

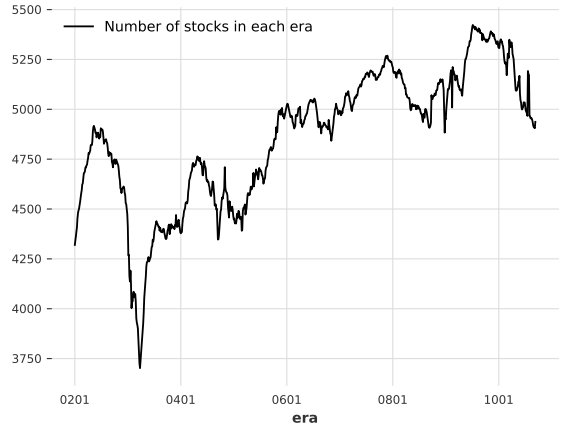


Figure 4.2: Number of stocks in each era from Era 201 to Era 1070 for v4.2 Numerai dataset.

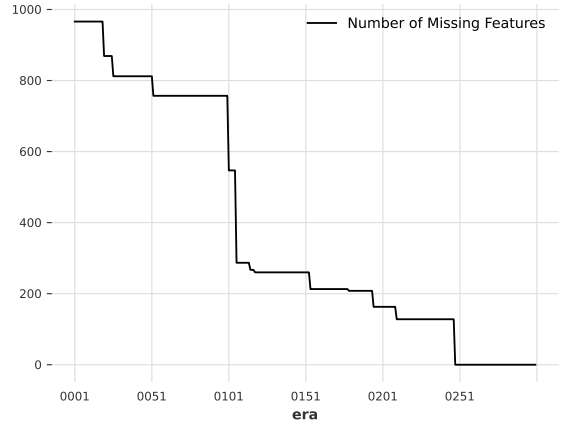


Figure 4.3: Number of features with missing values from Era 1 to Era 300 for v4.2 Numerai dataset.

In V4.2 dataset, some features have completely missing values up to Era 251. In Figure 4.3, 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 [28]. The Rain feature group consists of features that are created synthetically from features in other groups using information up to Era 585 [2].

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.

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

Scoring Function Numerai calculates a variant of Pearson correlation for all predictions in a single era t, as follows [168]: 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 The presence of regime changes is one of the reasons why machine learning trading strategies suffer from significant losses. 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 [169], 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 of more than 33%. Indeed, most risk-management metrics based on historical performances, such as Value-at-Risk (VaR) [170] would not be able to foresee this downturn.

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 in the competition also suffered from a large drawdown in the same period. In Figure 4.4, we show the Underwater (Drawdown) plot of the Numerai Meta Model. The drawdown between Era 1055–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 4.1, we report the performances of the Numerai Meta Model from Era 901

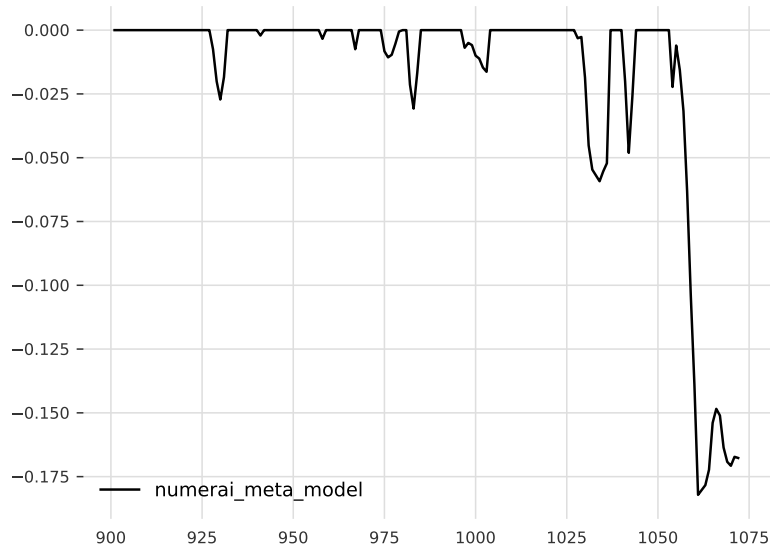


Figure 4.4: Underwater (Drawdown) plot of the Numerai Meta Model between Era 901 to Era 1070. A large drawdown is experienced by the model between Era 1055 to Era 1070.

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 (Eras 901-1050)	0.0207	1.0085	0.3491
Bear (Eras 1051-1070)	-0.0062	-0.3220	-0.0341

Table 4.1: Performances of Numerai Meta Model from Era 901 to Era 1070 under different market regimes.

4.6 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 already been used in the literature [160, 171, 172] and will serve here both as a baseline comparison and to guide some our choices in model type, training methods and hyperparameter selection. We note that 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.

4.6.1 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 12 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 4.3.1, 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. In particular, we train the following FT models:

- Exponential Moving Average factor-timing model: An EMA model (4.1) 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 12. These predictions are then scored using our portfolio metrics.

- Feature Transform factor-timing models (ST and RFT): 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 $\hat{z}_t \in \mathbb{R}^{N_t}$.

  We note that following recent research in high dimensional ridgeless regression [173, 160], 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 14, we describe the incremental learning procedure to train FT models.

Algorithm 14: 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 $\boldsymbol{\chi}_t \in \mathbb{R}^{2132}$ is obtained using the procedure described in Section 4.3.1

end

for $801 \leq i \leq 1070$ **do**

  Prepare training data by slicing the feature performances time series from Era 1 to $D_i - b$

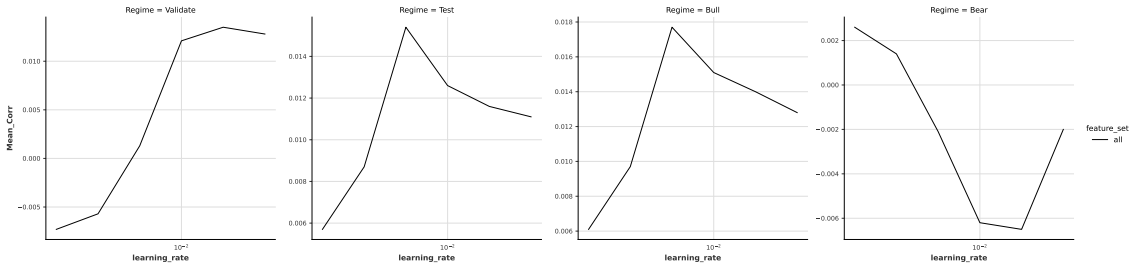
  Train Factor Timing models (EMA, ST. RFT).

  Get one-step ahead prediction y_{i+1} for era D_{i+1}

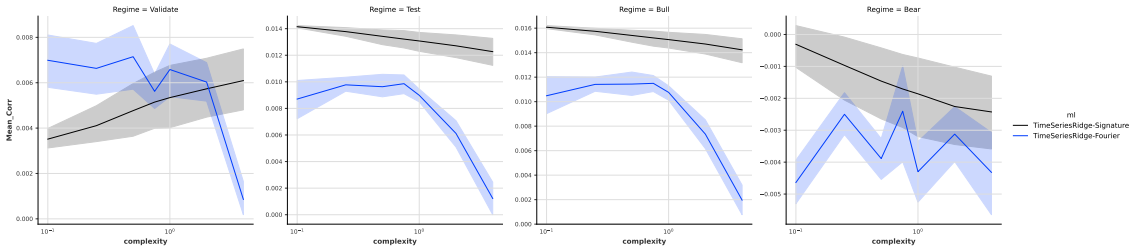
  Create factor timing predictions z_{i+1} using Algorithm 12

end

To optimise the 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 4.5(a) shows the performances of EMA models with different weight decays under different market regimes for weight decays $\alpha = [0.00125, 0.0025, 0.005, 0.01, 0.02, 0.04]$. Note that the best weight decay for Mean Corr in the validation period is 0.02, whereas it is 0.005 in the test period. This is another example of the effect of regime changes and why time-series cross-validation cannot always select the best weight decay for out-of-sample data. In particular, and as expected, models with a smaller decay constant perform better in Bear market but are the worst in Bull market. The key hyperparameter for the signture FT models (complexity) is explored in Figure 4.5(b) under different market regimes, averaged over 4 different random seeds. 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 the 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 complexity increases in the validation period but not in the test period. This suggests using more complex factor-timing models do not always give better results for FT models and reinforces the suggestion that hyperparameters selected based on time-series cross-validation might not be robust in out-of-sample data.



(a) EMA models



(b) Feature Transform factor-timing models

Figure 4.5: Mean Corr of EMA and Feature Transform factor-timing models under different market regimes

4.6.2 Benchmark IL model for XGBoost models

Two key hyperparameters for IL models are (i) training size, which for a temporal tabular dataset corresponds to the number of eras of data to be used in training; and (ii) retraining period, which governs how often the model is 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 sets of hyperparameters:

- a set found by grid search with fixed number of boosting rounds $B = 5000$ and learning rate $L = 0.01$, which we denote the Ansatz hyperparameter set

- a set provided by Numerai in their example Python script, denoted here the Numerai hyperparameter set.

In each model retrain, we 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 use 800 eras of data from Era 201 to train the models. The first retrain at Era 801 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, and as a reference comparison, we also create benchmark models that are not regularly retrained, which we call **Ansatz-Fixed** and **Numerai-Fixed** respectively.

The details on the procedure to create hyperparameter sets and model training are described in Sections 4.9.2 and 4.9.2 in the 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). We show in SI Section 4.9.2 that this sampling method does not deteriorate model performance, while reducing computational costs by half compared to training using all data.

Finally, in order to manage the computational constraints, we produce regular samples of the data eras in the training period such that only 25% of the data eras is used 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, and similarly for the other 3 models. We call this procedure **regular era sampling**.

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 the model once per year. We do not retrain models with $B \geq 10000$ due to computational limitations. The learning rates of the model are determined using the Ansatz formula $L = \frac{50}{B}$, which is explained in detail in Section 4.9.2 in SI.

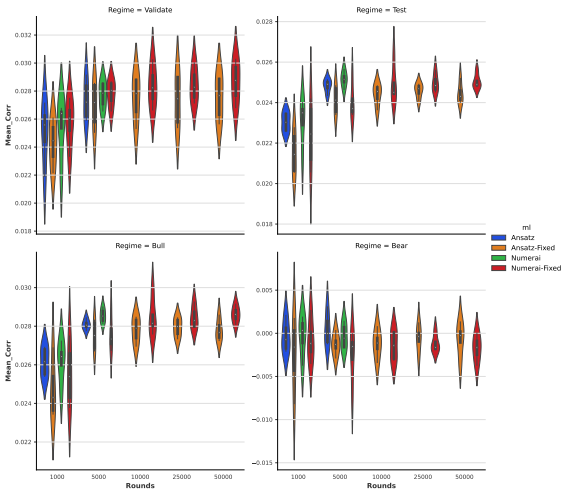
We report performances of the benchmark models from Era 801 to Era 1070 according to the following regimes:

- Validation: Era 801 - Era 885
- Test: Era 901 - Era 1070
  - Bull: Era 901 - Era 1050
  - Bear: Era 1051 - Era 1070

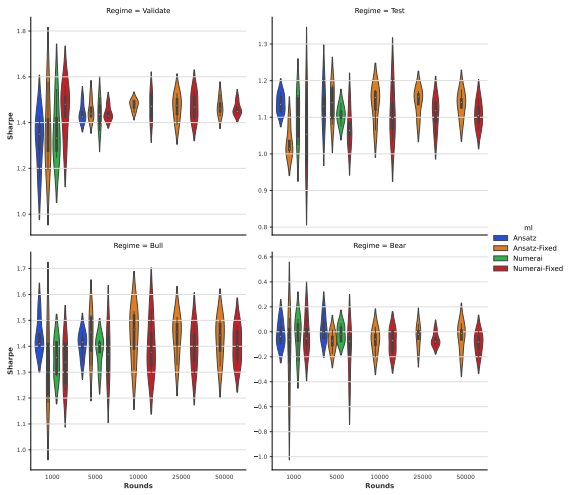
Figure 4.6 shows the performance of the benchmark models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$. Model performances for models with $B \geq 5000$ are not significantly different in both validation and test period. Within different market regimes in the test period, there are also no significant differences in model performance for $B \geq 10000$.

Furthermore, there are no significant differences between each pair of models using the two different hyperparameter sets Ansatz and Numerai for a fixed data sampling method (regular retrain vs no retrain) in both the validation and test period. Yet as the number of boosting rounds increases, the performance differences between models with different hyperparameters narrows. This suggests that the differences between models performances are more due to differences between data sampling schemes used and to a lesser extent due to the hyperparameters.

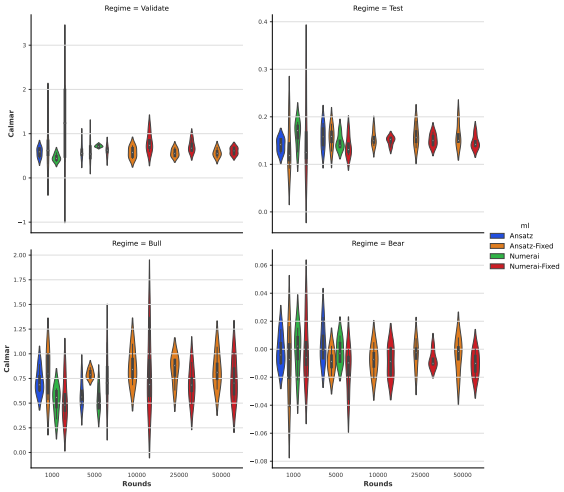
To select our benchmark model hyperparameters, we consider both model performance and computational resources. Despite Numerai models having a slightly better Mean Corr than Ansatz models, they suffer from higher computational time and memory costs (explained in detail in SI Section 4.9.2)



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 4.6: 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

and do not exhibit an improvement in Mean Corr in the validation period. Further, models with Ansatz hyperparameters also have a lower variance than those with Numerai hyperparameters across different risk metrics (Mean Corr, Sharpe ratio). Given their similar performance and characteristics, we select the Ansatz hyperparameter set to train different deep IL models in the next section.

How to measure the similarity between two GBDT models To measure the similarity between two GBDT models, we need to consider the overall structure similarity between two models also using feature importance, as this 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 only correlation between predictions because predictions that are similar 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 to drifts in the data.

Definition (Structural Similarity of GBDT models)**.** For simplicity, a correlation based measure is used here to measure the overall similarity of two GBDT models, as follows 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

$$r_A = \text{rank}(R_A) \in [0, 1]^M \tag{4.2}$$

$$r_B = \text{rank}(R_B) \in [0, 1]^M \tag{4.3}$$

$$\mathcal{S}(A, B) = \mathbf{Corr}(r_A, r_B) \tag{4.4}$$

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

Note that this measure considers the averaged contribution of each feature towards the model and ignores the interaction between features.

Differences between benchmark models

To understand the differences between models trained with different hyperparameters, we use the structural similarity measure (4.4) to understand the overall structural similarity between the Ansatz and Numerai benchmark models.

In Figure 4.7, we show the temporal correlation structure of the Ansatz and Numerai benchmark models with sizes $B = 1000, 5000$. As expected, the correlation between the retrained models decreases as the time gap between model retrains increases. Smaller models ($B = 1000$) are less correlated with each other than larger models ($B = 5000$), which can have $\mathcal{S} > 0.9$ even after 150 eras. This observation supports our choice not to 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.

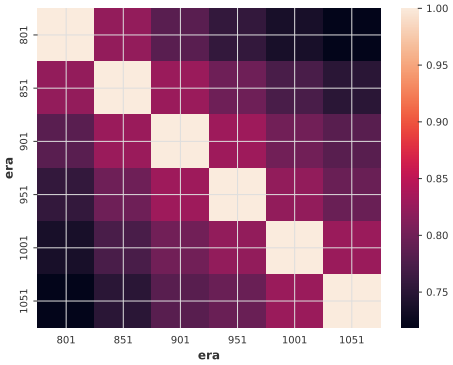
Figure 4.8 shows the correlation structure of benchmark models of different sizes at Era 801, the first model training time. For both Ansatz and Numerai hyperparameters, models with sizes $B \geq 5000$ are highly correlated. Cross-correlation between Ansatz and Numerai models of the same size is lower 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 that hyperparameter optimisation is not necessary for large GBDT models so that we may pick any reasonable hyperparameter set (e.g., Ansatz) based on computational requirements.

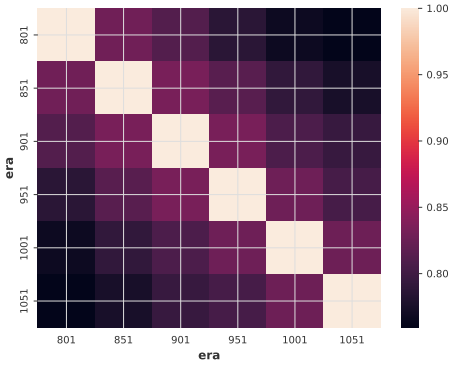
Considering the fact model performances do not significantly improve beyond $B = 5000$, we conclude it is not necessary to train *single* models with size $B > 5000$ as it consumes more computational resources while not providing meaningful gain in model performance. It is better instead to allocate the computational resources to train in parallel an ensemble of models with size $B \leq 5000$.

4.7 Deep IL XGBoost Models

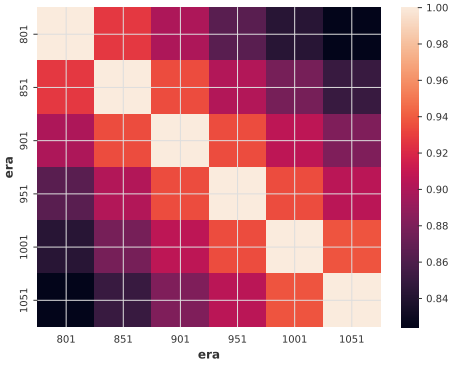
We now deploy the full deep IL model with dynamic ensembling, in which models trained with different sampling schemes and hyperparameters are combined dynamically to create better models. This is inspired by our previous work on dynamic forecasting in financial data [151] and by models used in weather forecasting [174], where ensemble forecasting has been used to improve robustness of predictions.



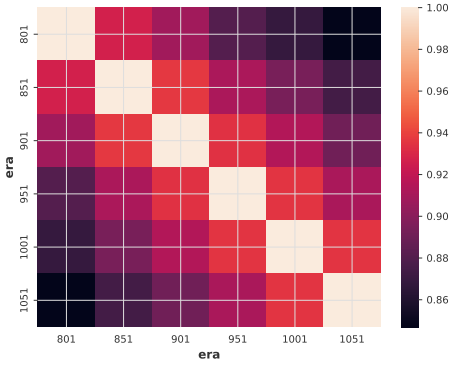
(a) Correlation between models with $B = 1000$ and Ansatz hyperparameters



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

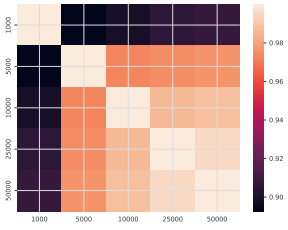


(c) Correlation between models with $B = 5000$ and Ansatz hyperparameters

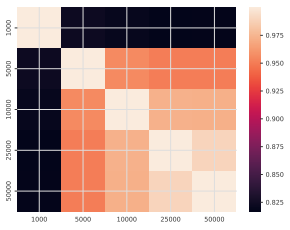


(d) Correlation between models with $B = 5000$ and Numerai hyperparameters

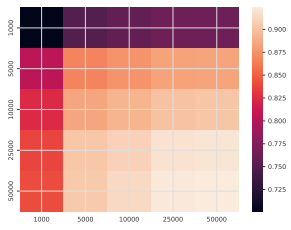
Figure 4.7: Temporal correlation structure of benchmark models from Eras 801 to Era 1051.



(a) Correlation between models with Ansatz hyperparameters



(b) Correlation between models with Numerai hyperparameters



(c) Cross Correlation between Ansatz and Numerai models

Figure 4.8: Correlation structure of benchmark models of different sizes at Era 801

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.

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 exploredifferent sampling strategies to create diversified base models. In particular, we study model ensembles created with different (1) training sizes, (2) learning rates, (3) targets and (4) feature sets. 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.

The deep IL models used a 2-layer model structure. The Layer 1 models are XGBoost models trained with different hyperparameters and settings described below. The Layer 2 models N_k, $1 \leq k \leq 2$ are chosen 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.

The purpose of Layer 2 models is to refine predictions obtained in Layer 1. By combining predictions at individual observation (stock) level instead of model level, this approach is more flexible than the dynamic model selection used in Chapter 3.

4.7.1 Ensemble strategies based on data sampling

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 historical regimes, but also have the risk of including data no longer relevant. If less data is used, the training data can adapt more quickly to concept drift, but can 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 15, the maximum training set size of Layer 1 models is 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 is determined by the Ansatz formula $L = \frac{50}{B}$, using the scaled number of boosting rounds.

In Figure 4.9, we compare the performances of the two Layer 2 models (Elastic Net, Equal Weighted) with the benchmark Ansatz model of $B = 5000$. The Equal Weighted model over all possible training set sizes achieves a higher Mean Corr than the benchmark model in the test period, yet there is improvement in the Bull market but not in the Bear market. The 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.

Algorithm 15: 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

4.7.2 Ensemble strategies based on different learning strategies

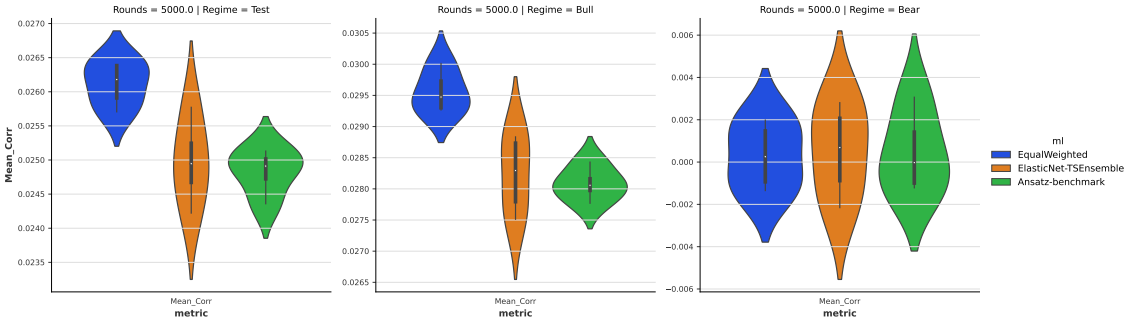
Learning Rate model ensemble (Complexity ensemble)

Recent research suggests that features are learnt with different speeds within a neural network [175, 176]. Inspired by this idea, we combine GBDT models with different learning rates to learn models capturing both fast and slowing features.

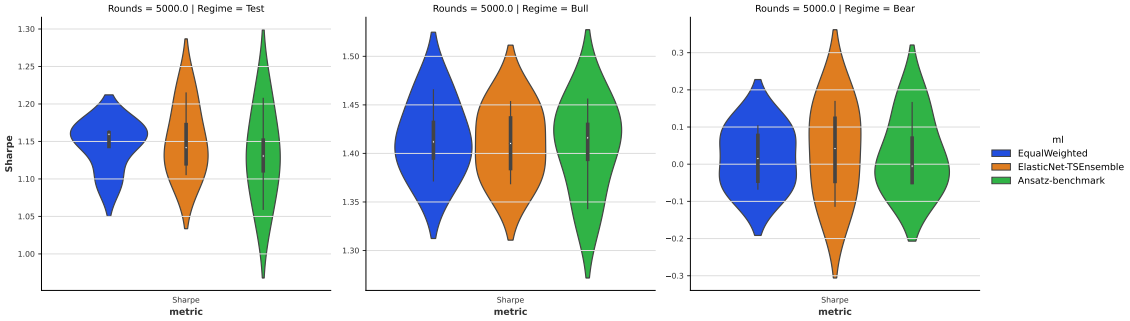
In Algorithm16, we combine XGBoost models with 5 different learning rates. For a given number of boosting rounds B, in addition to training the model of size B as above, we train two larger models of size $2B$ and $4B$ and two smaller models of size $\frac{B}{2}$ and $\frac{B}{4}$ where the learning 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 Algorithm16 are the same as those used in Algorithm 15. Since 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.

We run Algorithm 16 for $B = 5000$ and performances for the two Layer 2 models are shown in Figure 4.10, compared with the Ansatz benchmark model with $B = 5000$. Both the Equal Weighted and Elastic Net models 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 in the Bear market.

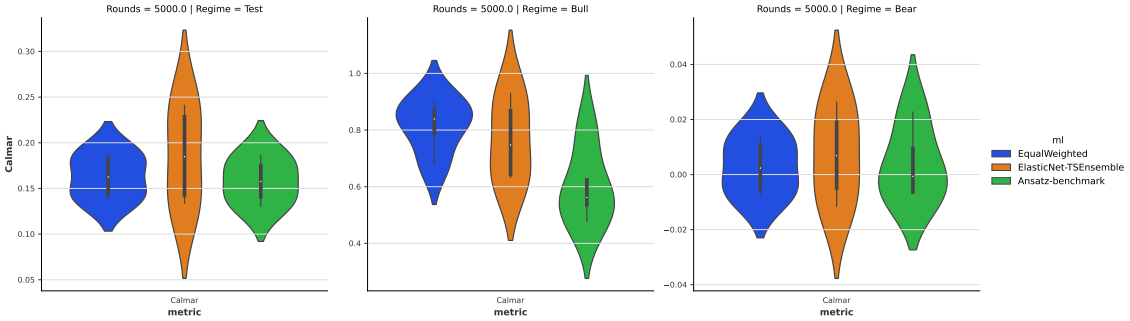
In Figure 4.19 in SI, we show the learning curves of the 5 Layer 1 XGBoost models with different learning rates and the corresponding number of boosting rounds (1250,2500,5000,10000,20000). Although



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 4.9: 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 $B = 5000$.

larger models perform slightly better than smaller models in the validation and test period, there are no significant differences in model performances in the Bear market. Therefore, there is no single optimal complexity across all regimes. This observation supports our proposed use of deep IL to combine the strength of models with different complexity (learning rates) so that the ensemble model is more robust.

Algorithm 16: 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

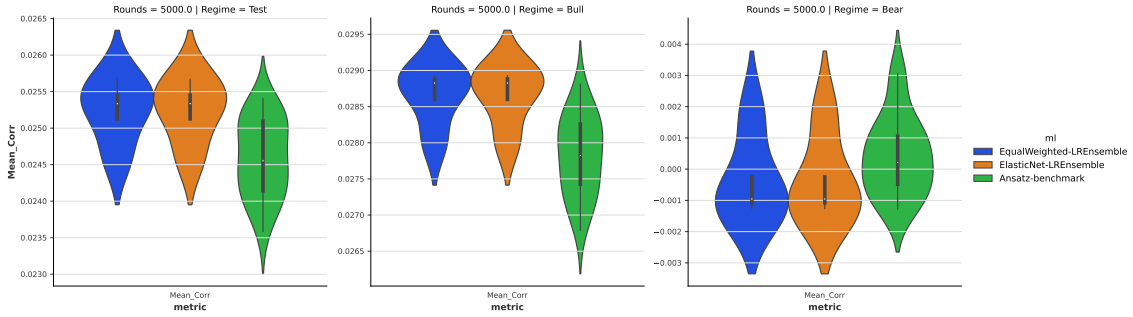
4.7.3 Ensemble strategies based on different targets

Feature projection was used in Chapter 3 to reduce drawdown of trading strategies. In the V4.2 dataset [28], 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 lower risks (Max Drawdown and Volatility). The overall risk profile is improved even the portfolio return is reduced.

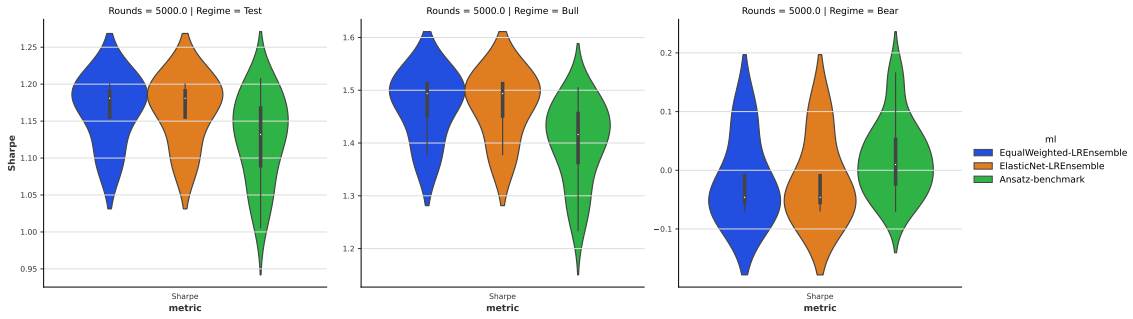
Model ensemble with different targets and learning rates

In Algorithm17, we combine XGBoost models trained with the five different targets using different learning rates as in Algorithm 16. In total, we train 25 Layer 1 XGBoost models to be combined in Layer 2. The Layer 2 models N_k used in Algorithm17 are the same as those used in Algorithm 15.

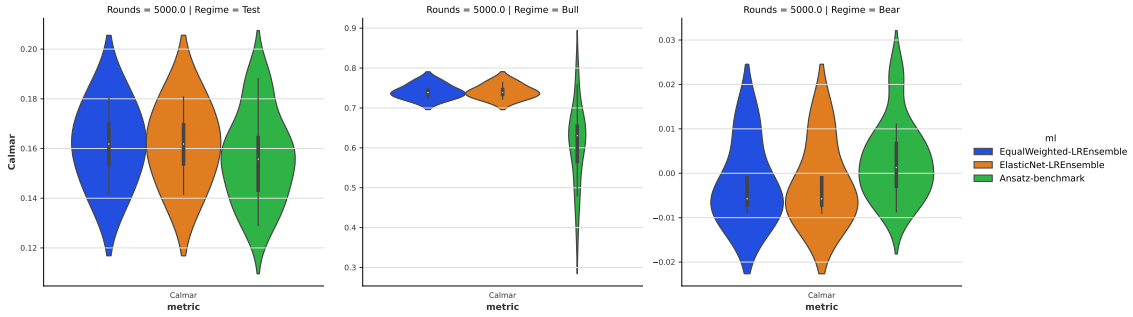
In Figure 4.11, we compare the performances of the two Layer 2 models (Elastic Net, Equal Weighted) against the benchmark Ansatz model of $B = 5000$. The Equal Weighted model over all 25 Layer 1 XGBoost models with different targets over different learning rates achieves a higher Sharpe and Calmar ratio 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



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 4.10: 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.

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 performs 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 17: 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

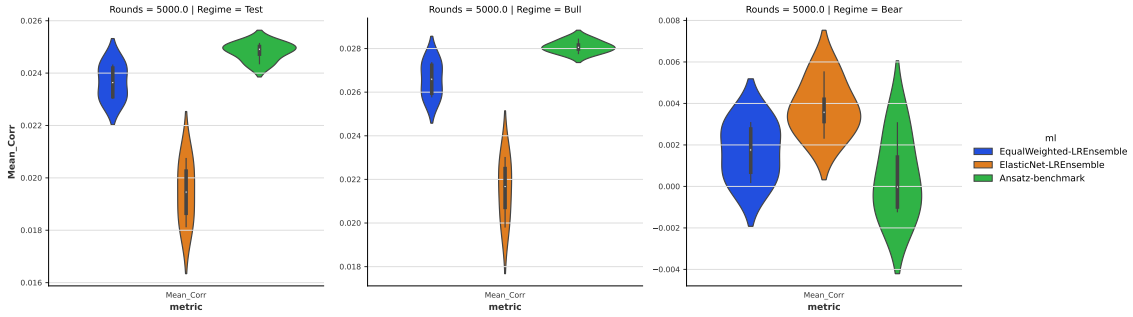
end

4.7.4 Ensemble strategies based on feature sampling

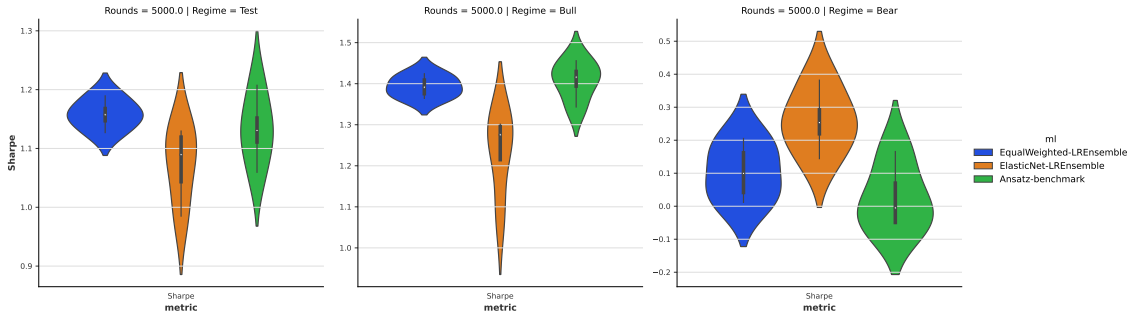
Feature Sets model ensemble

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, as some features will never be used in the overall model rather than simply missing in some trees. It also lowers computational requirements of models as we do not have to fit all the data to the model. Here, we study Jackknife sampling [177] among other sampling techniques to build diversified models suitable for ensembling.

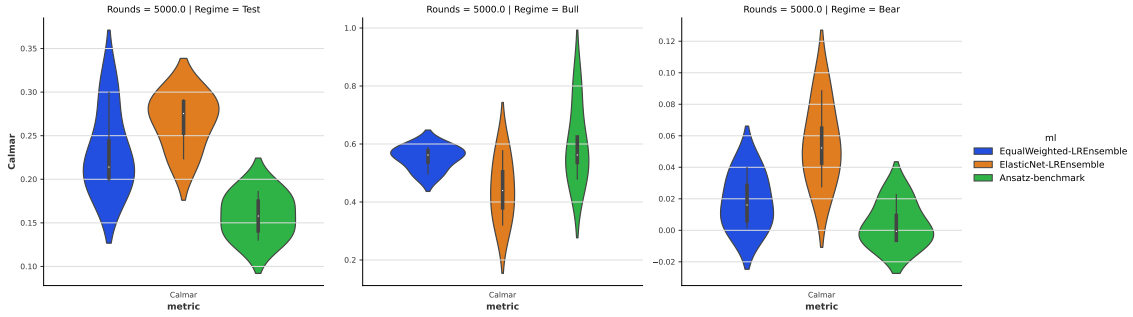
The feature group labels in Numerai V4.2 dataset can be considered as a way of feature clustering using domain knowledge. Instead of analysing the high-dimensional temporal correlation structure of the features by clustering or dimensionality reduction methods, we use the labels provided by Numerai,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



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 4.11: 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.

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 the remaining 9 groups to form the Jackknife feature sets $\mathcal{F}_1 \ldots \mathcal{F}_{10}$. The Jackknife feature sets are then used to train XGBoost models, using the procedure described in Algorithm18. The Layer 2 models N_k used in Algorithm18 are the same as those used in Algorithm15.

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, as described in Algorithm19; and (ii) benchmark XGBoost models trained with all the features using Ansatz hyperparameters. The Layer 2 models N_k used in Algorithm19 are the same as those used in Algorithm18. The reason to use (i) as a benchmark is to calibrate if feature group labels offer information that is better than random in separating the features into groups representing different signal sources, thus creating information barriers between models so that they are forced to learn rules that are different from each

other. This would reduce correlation between predictions. The reason to use (ii) as a benchmark is to check if any form of feature selection is beneficial to model performance at all.

Algorithm 18: 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 19: 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 $b_1 = 15$, $b_2 = 6$

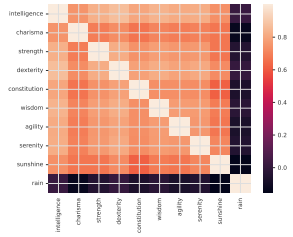
Set starting Era $D = 801$
Set Ansatz learning rate $L = \frac{50}{B}$
for $1 \leq i \leq 6$ **do**
  Set $D_1 = D + (i-1)T$
  Prepare training data from Era 201 to $D_1 - b_1 + (i-1)T$
  for $1 \leq j \leq 10$ **do**
    Train Layer 1 XGBoost models M_j^i, with 50% of the 2132 features selected by random 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

We run these two algorithms for $B = 5000$. In Figure 4.13 we show the performances of the four Layer 2 models from Jackknife and random feature sampling against the benchmark Ansatz model with $B = 5000$, which is also regularly retrained. The Layer 2 models from Jackknife sampling have a higher

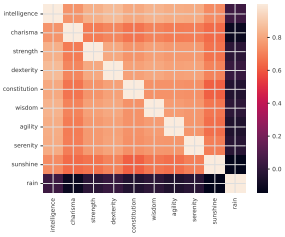
Mean Corr and Sharpe ratio than the models from random sampling and the benchmark model in both the validation and test period. The Layer 2 models using random sampling have comparable Mean Corr and Sharpe ratio with the benchmark model in both the validation and test period. Within models using Jackknife sampling, there are no significant differences between the Equal Weighted and Elastic Net model. However, for models using random sampling, Elastic Net model underperformed relative to the 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 them.

In Figure 4.12, 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.

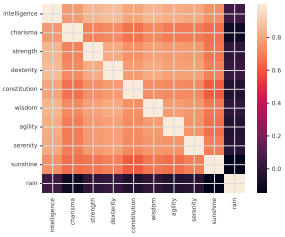
While highly similar models offer limited diversification benefits in model ensembling, uncorrelated models created by random failed to generate better predictions. In conclusion, using domain knowledge about the data generation process, we can create models that are not over-similar to each other which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



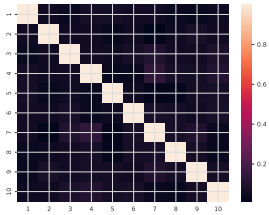
(a) Structural similarity of models with Jackknife feature sampling at Era 801



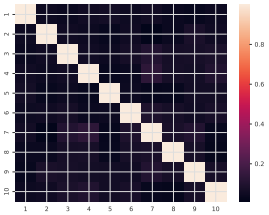
(b) Structural similarity of models with Jackknife feature sampling at Era 901



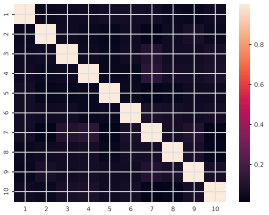
(c) Structural similarity of models with Jackknife feature sampling at Era 1001



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

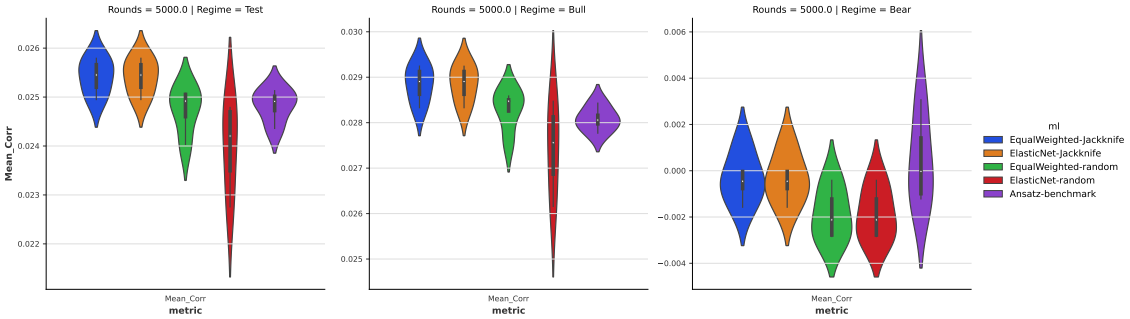


(e) Structural similarity of models with random feature sampling at Era 901

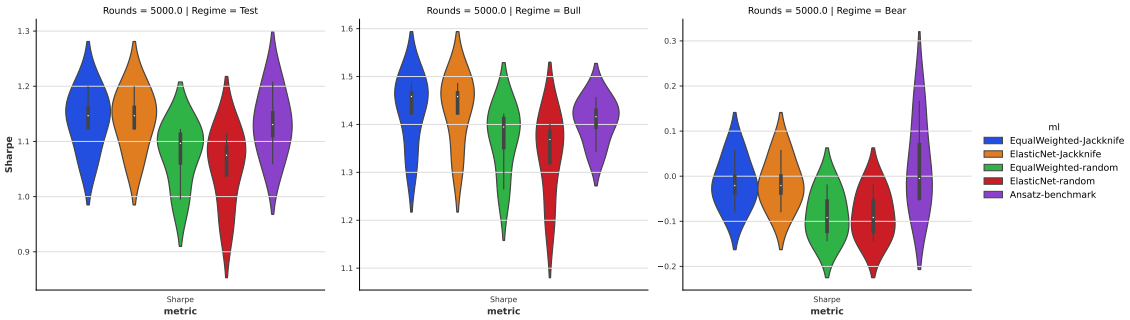


(f) Structural similarity of models with random feature sampling at Era 1001

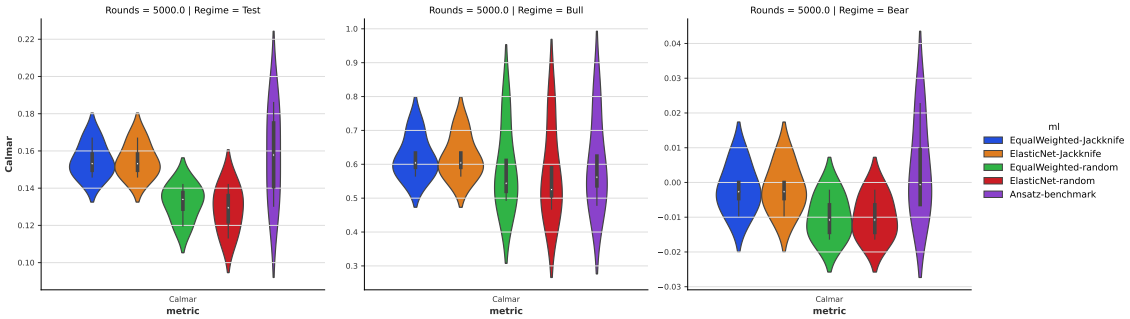
Figure 4.12: Structural similarity of models with Jackknife and random feature sampling at Era 801, 901, 1001



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 4.13: 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.

4.7.5 Dynamic Hedging based on model variances

Recent research suggests that disagreement between investors is an indicative signal of stock returns [178]. In particular, under the Bull market, stocks that have the highest degree of disagreement between investors will under-perform relative to 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 Algorithm20, 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 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 then 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, is

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 performance of the Tail Risk strategy.

In Table 4.2, we compare the dynamic hedged model from deep IL XGBoost ensemble using feature set Jackknife and the dynamic hedging strategy described above, with the example model provided by Numerai. To note, the example model is 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 its complementary nature. The Dynamic hedged model achieves comparable Mean Corr with the example model provided by Numerai. The Sharpe ratio is 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 is much smoother for the Dynamic hedged model, as shown in Figure 4.15. We have checked 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 4.6,4.9,4.8,4.7 in SI. The results show that dynamic hedged models from these ensembles are inferior to the above, as they have a lower Mean Corr and Sharpe ratio.

Algorithm 20: Model Disagreement

Input: At era t: predicted values $\hat{y}_k^t \in \mathbb{R}^{N_t}$ from Layer 1 models $\mathcal{M}_k$, $1 \leq k \leq K$

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)

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

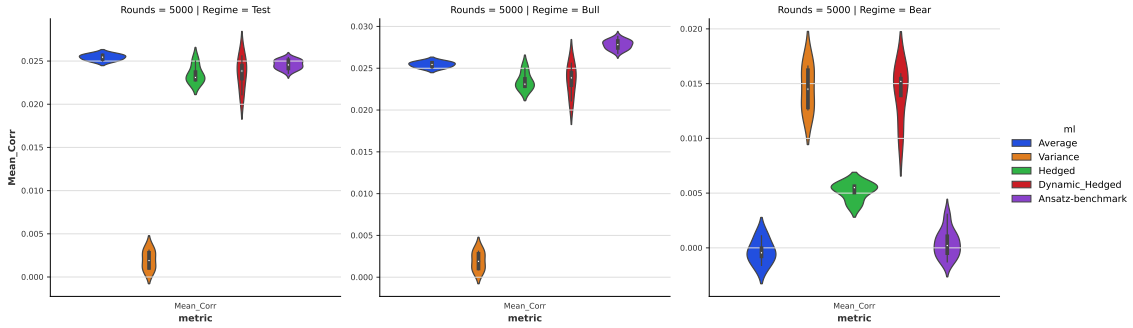
Calculate static hedged model predictions $\hat{h}_t = 0.6\hat{y}_{mean}^t + 0.4\hat{y}_{sd}^t$

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 4.5.

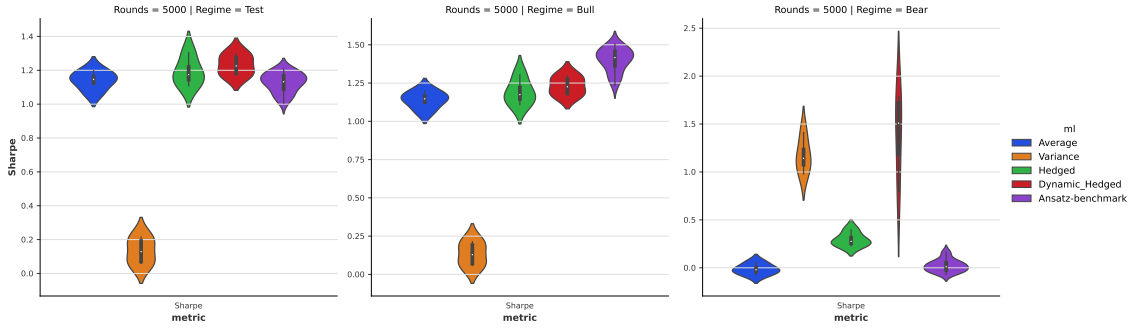
if $\bar{\rho}_t \geq 0$ **then**
  Set Hedge ratio $h = 0.6$
end
else
  Set Hedge ratio $h = 0$
end

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

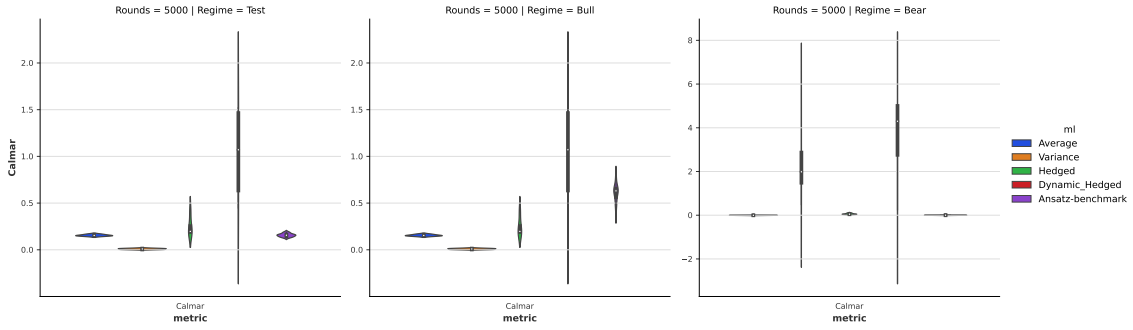
Model ensembles created based on Jackknife feature set sampling use knowledge about the dataset and thus offer a better approximation of disagreement between investors. Therefore, the Tail Risk model created based on the variance between models trained with Jackknife feature set sampling is the most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



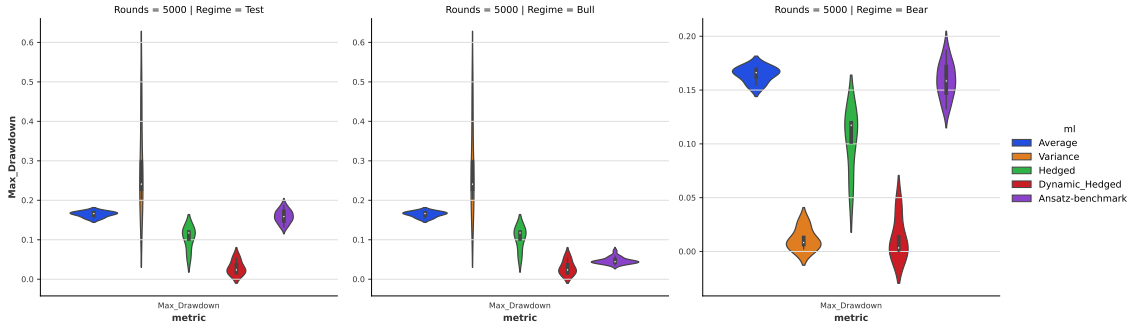
(a) Mean Corr



(b) Sharpe



(c) Calmar



(d) Max Drawdown

Figure 4.14: 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.

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 4.2: 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.

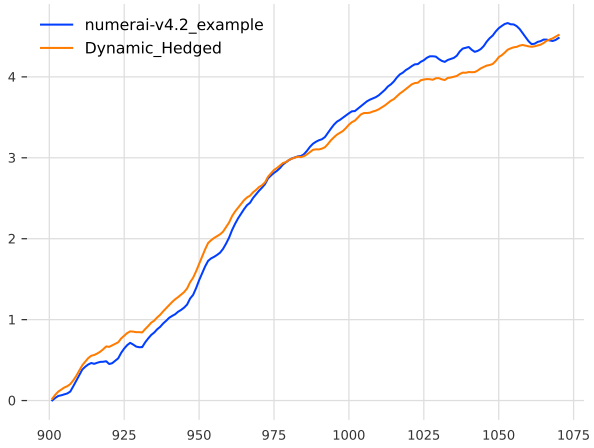


Figure 4.15: The portfolio return curve of the Dynamic Hedge model based on feature set Jackknife sampling and V4.2 Example Model from Era 901 to Era 1070

4.8 Conclusion

In this study, both traditional tabular and factor-timing models have been 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.

GBDT are robust ML models We found that GBDT models performed best for the Numerai datasets, agreeing with the findings of [16], 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 performance, slowly converging to the theoretical optimal.

GBDT models are also highly scalable and have robust performance over slightly perturbed hyperparameters. The larger the GBDT model, the smaller the effect of model hyperparameters on the learning process and model performances. GBDT models are numerically stable, without the convergence issues that commonly plague the training of neural networks.

Feature and Data Sampling Data management and forgetting mechanism is an integrated part of an IL pipeline [1] to build robust prediction models on a data stream. The impact of data sampling methods is usually over-looked in most quantitative finance research and even in hedge funds [152]. Data and feature sampling methods can have significant effects on model performances.

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

Feature sampling can also 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 [179], 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.

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 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 is 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 [180, 181] 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 simplest form of deep learning models, MLP in this paper. Recent research suggests regularisation techniques [23] can improve performances of neural networks models over a wide range of network architecture. Further work can be done to investigate if careful design of the model training process with suitable regularisation can improve the scalability and model performances.

4.9 Supplementary Information

4.9.1 Algorithms of different benchmark machine learning models studied

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 $X_t = (X_t^1, X_t^2, \ldots, X_t^d)$ 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 $i_1, \ldots, i_k$ is defined as

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

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

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

An alternative definition of signature as the response of an exponential nonlinear system is given in [159].

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 [158, 159]. 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 [182]. 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 [158, 159]. 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 [183]. This can be considered as a version of the universal approximation theorem [163] 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 [184], 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 21 can be used to obtain paths, which are slices of time series with different lookback windows. Random Signature transforms 22 can then be used to compute the signature of the path, which summarises the information of the time series.

Algorithm 21: 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 22: 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 $s_t = (s_t^1, \ldots, s_t^p)$

Random Fourier Transforms

Random Fourier Transforms are used in [160] 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 23. The key idea is to approximate a mixture model of Gaussian kernels with trigonometric functions [161].

Algorithm 23: Random Fourier Transform [160]

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

Gradient Boosting Models

XGBoost Implementation XGBoost [11] 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.

Deep Learning Models

Training process PyTorch Lightning [185] 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 24 shows how to create the feature engineering part of the network.

Funnel architecture, as used in [186] is an effective way to define the neuron sizes in a network for different input feature sizes. Algorithm 25 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 9 (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 24: 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 25: 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.

4.9.2 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 [17]. 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 4.9.2 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 4.9.2. 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 [187] for their example model based on LightGBM [12]. 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 4.26, 4.27, and 4.28 in SI, learning curves for Mean Corr and Sharpe ratio of the benchmark XGBoost models are shown under different market regimes.

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 Algorithm26.

Algorithm 26: 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 4.21, 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 4.22 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.

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 [16, 15, 14], 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 [188, 189], either the number of boosting rounds and/or the learning rate is fixed and then the other hyperparameter is optimised. Recent research [180] 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 Algorithm27 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 27: 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 4.23, 4.24 and 4.25, 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.

Selecting ML models for Temporal Tabular Datasets

For different tabular models introduced in section 4.3.2, 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 4.3.2.

- 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 4.3 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	**0.0235** ± 0.0001	**1.1344** ± 0.0119	**0.2692** ± 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 4.3: 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 4.4 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 4.5 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 [190], 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 [14], 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 [15, 14]. 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 [17] 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 4.4: 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)

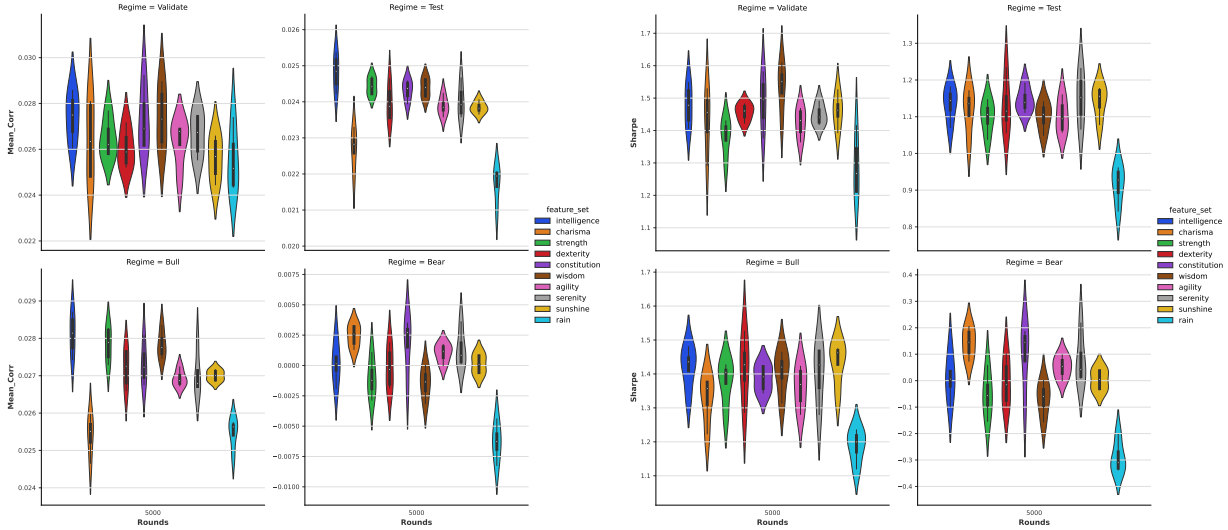
Max Depth	L2-reg	L1-reg	Mean Corr	Sharpe	Calmar
4	0.0	0.0	**0.0276** ± 0.0007	**1.3829** ± 0.0515	0.5305 ± 0.1077
		0.001	0.0274 ± 0.0006	1.3764 ± 0.0485	0.5161 ± 0.1213
		0.1	0.0268 ± 0.0008	1.3283 ± 0.0627	0.4224 ± 0.0937
	0.001	0.0	**0.0276** ± 0.0007	**1.3829** ± 0.0515	0.5305 ± 0.1077
		0.001	0.0274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

Table 4.5: XGBoost models with different L1 and L2 regularisation with fixed data and feature subsampling ratios of 75% between 2014-07-04 (Era 601) and 2018-04-27 (Era 800)

4.9.3 Additional Results

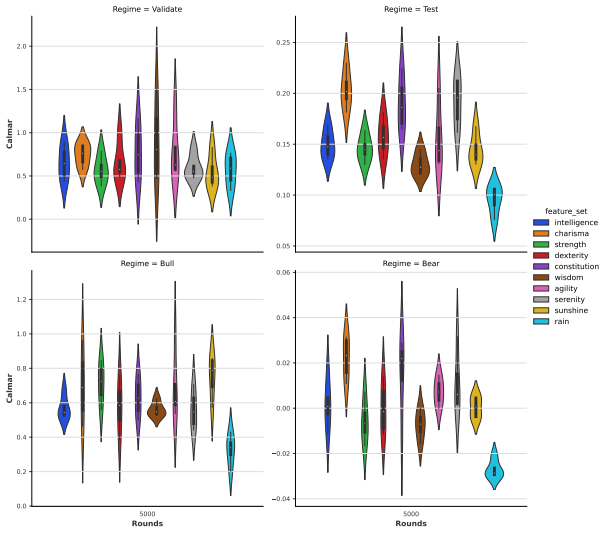
Regime	Strategy	Mean Corr	Sharpe	Max Drawdown
Test	Example Model	0.0264	0.9626	0.2608
	Baseline Model	0.0257	1.0983	0.1839
	Tail Risk Model	0.0022	0.1607	0.2161
	Static Hedged Model	0.0176	1.0507	0.0586
	Dynamic Hedged Model	0.0224	1.2378	0.0415
Bull	Example Model	0.0307	1.2512	0.0693
	Baseline Model	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 4.6: 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.



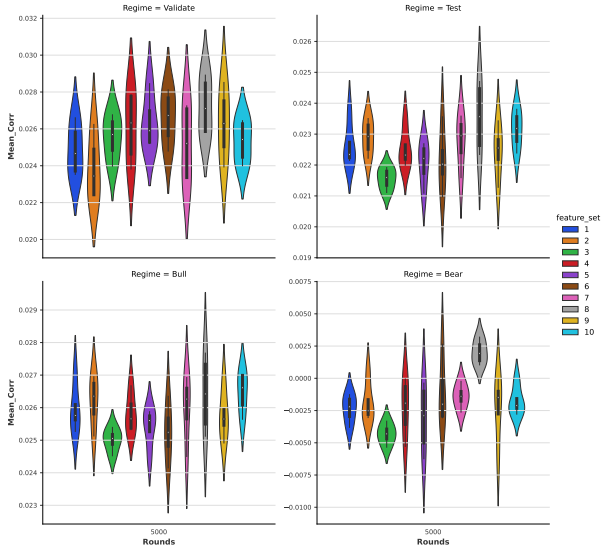
(a) Mean Corr

(b) Sharpe

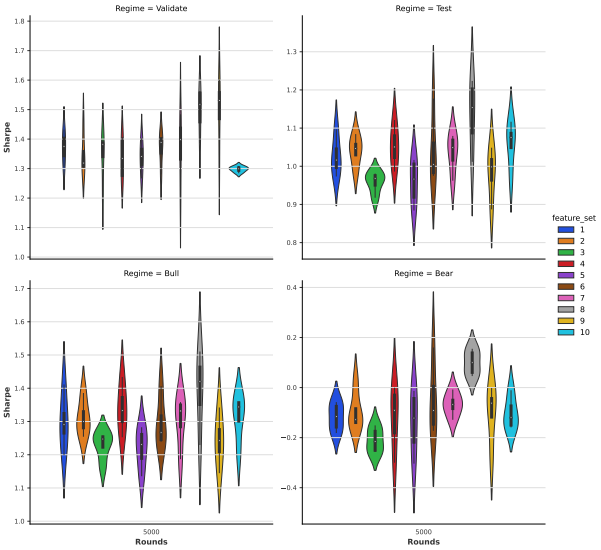


(c) Calmar

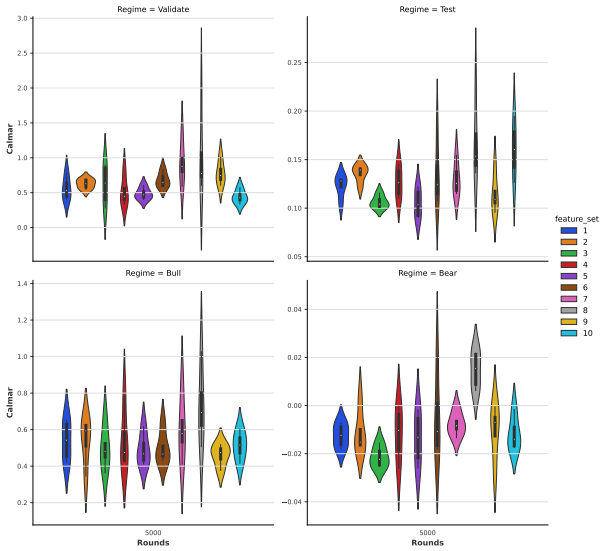
Figure 4.16: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with Jackknife feature sampling under different market regimes.



(a) Mean Corr

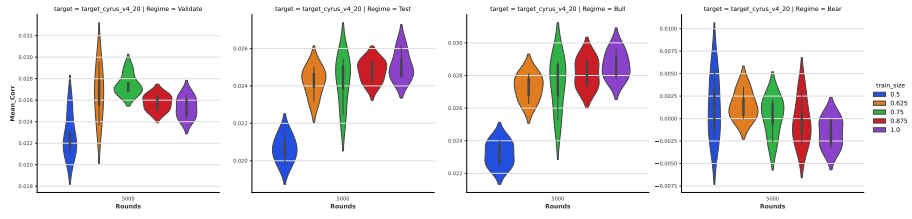


(b) Sharpe

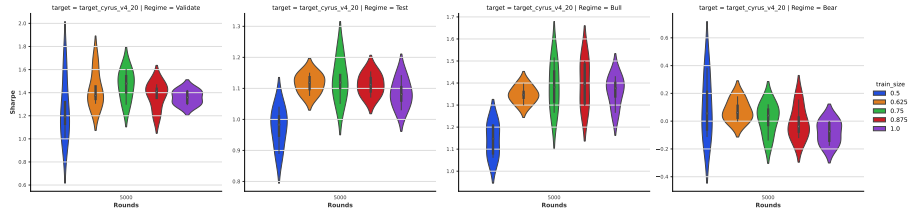


(c) Calmar

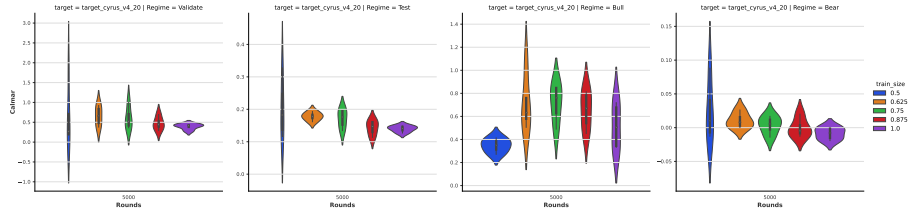
Figure 4.17: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with random feature sampling under different market regimes.



(a) Mean Corr

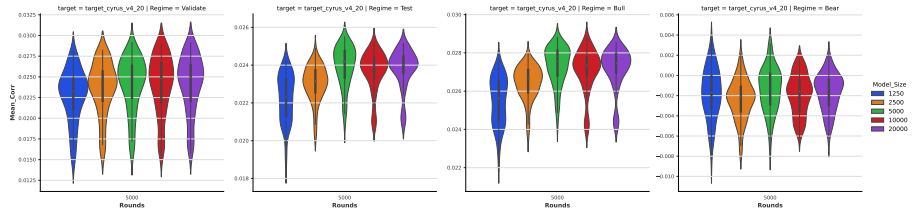


(b) Sharpe

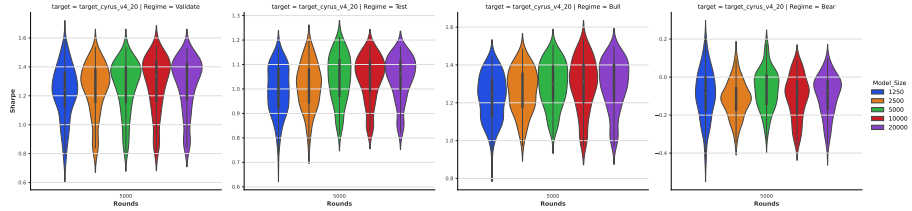


(c) Calmar

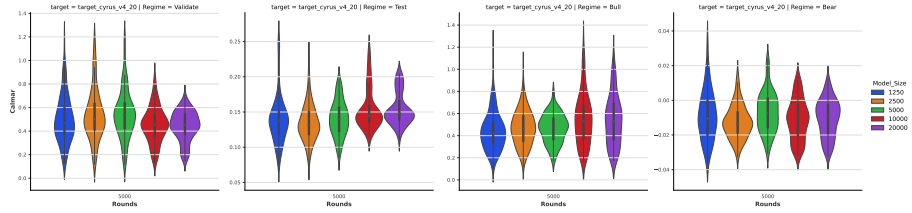
Figure 4.18: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with different training sizes under different market regimes.



(a) Mean Corr

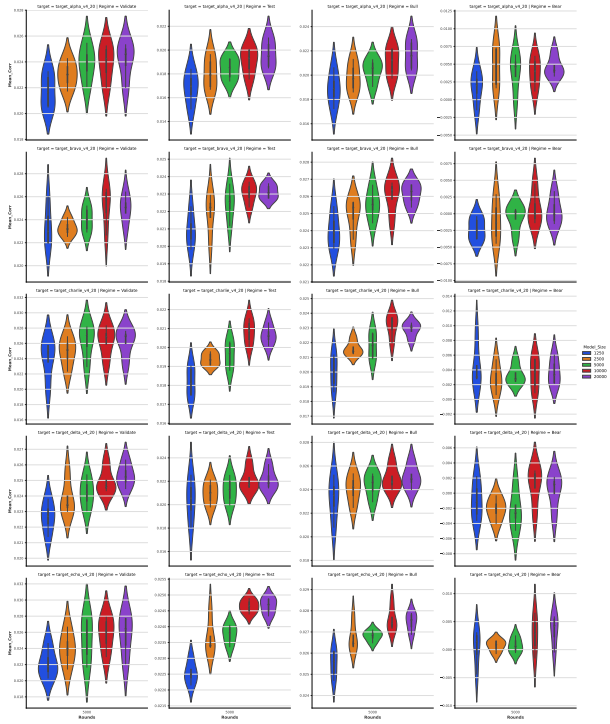


(b) Sharpe

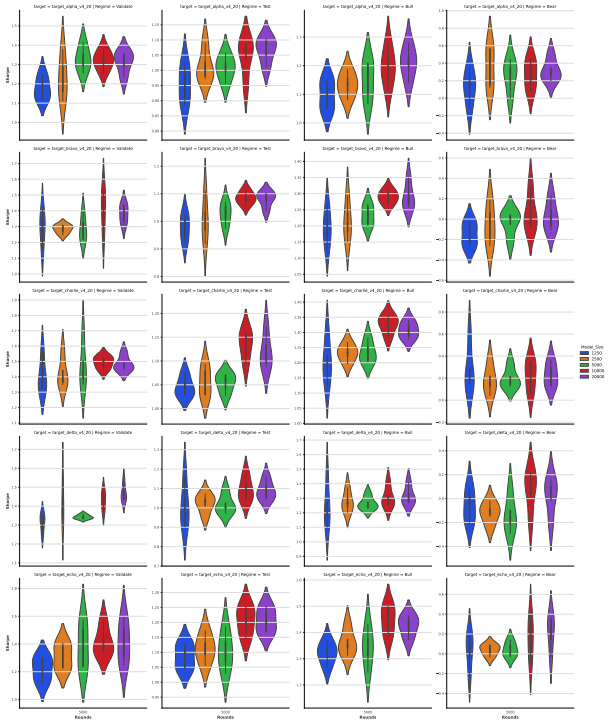


(c) Calmar

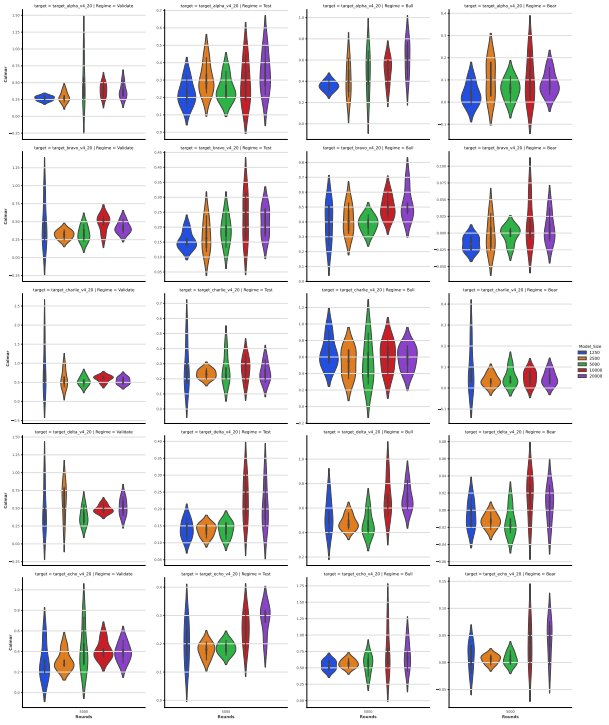
Figure 4.19: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with different learning rates under different market regimes.



(a) Mean Corr

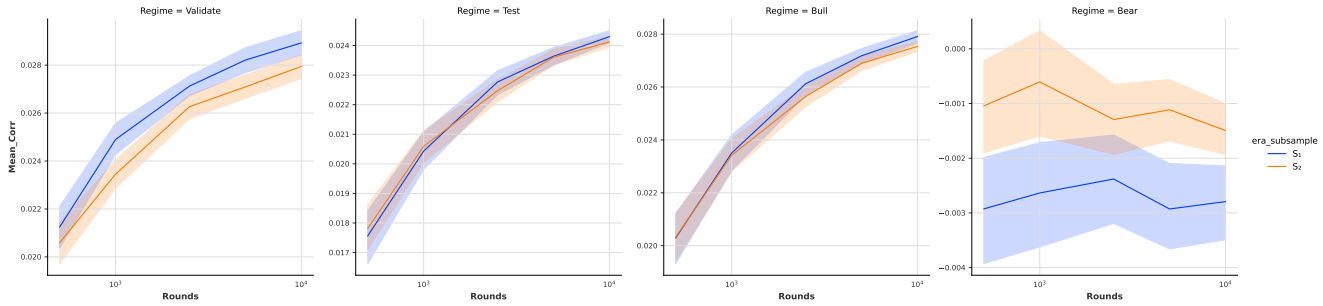


(b) Sharpe

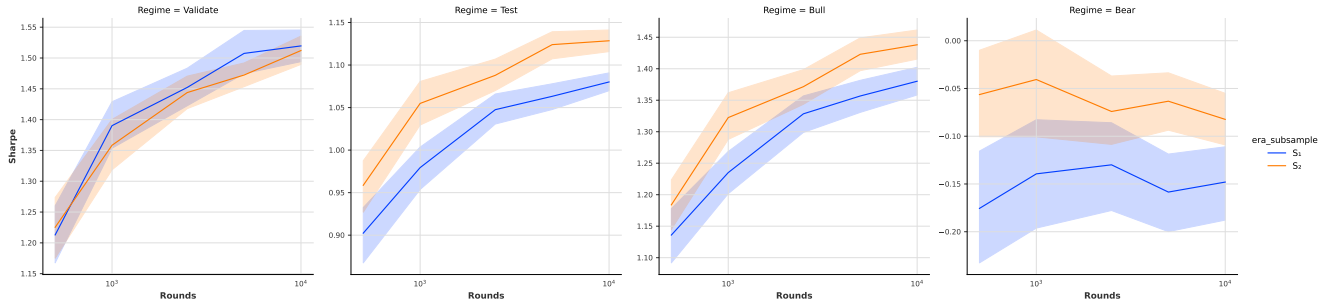


(c) Calmar

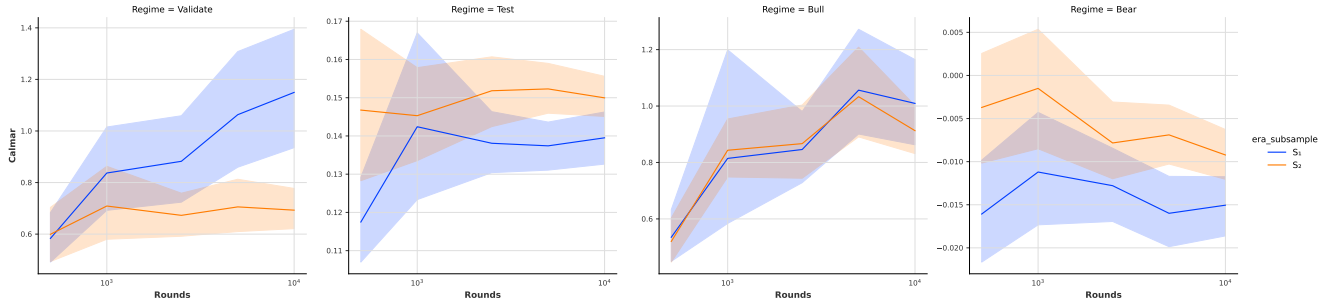
Figure 4.20: Performances, (a) Mean Corr, (b) Sharpe ratio, and (c) Calmar ratio of the deep IL XGBoost models with different targets and learning rates under different market regimes.



(a) Mean Corr

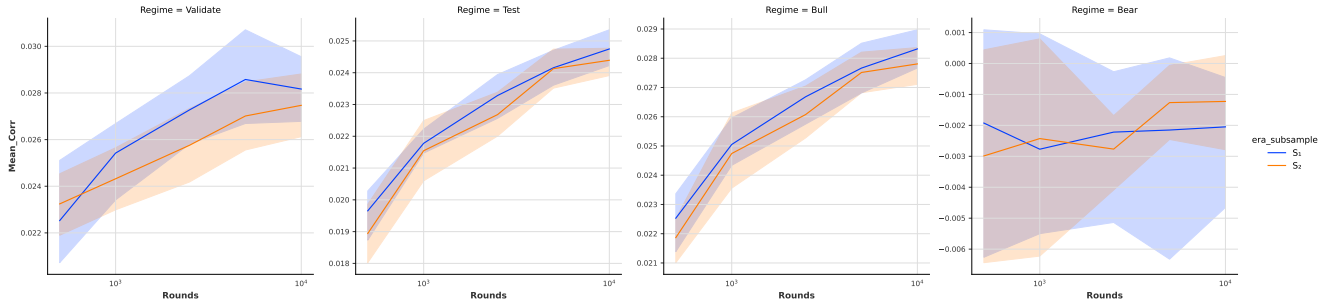


(b) Sharpe

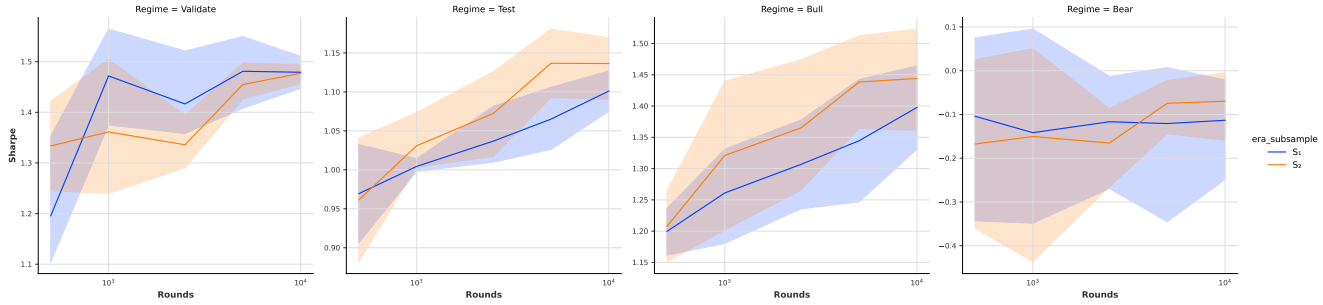


(c) Calmar

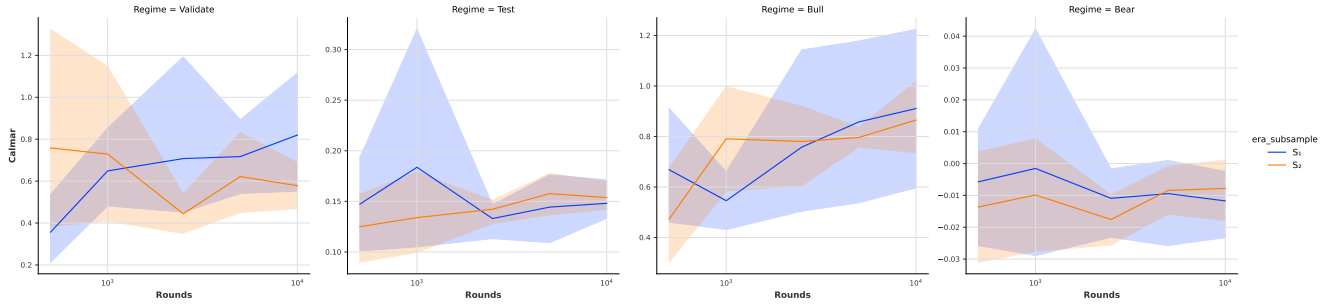
Figure 4.21: Comparing the performances of two data sampling schemes S_1 and S_2 with different number of boosting rounds $B = 500, 1000, 2500, 5000$ for risk metrics (a) Mean Corr, (b) Sharpe ratio, (c) Calmar ratio, under different market regimes, over 10 different hyperparameter settings



(a) Mean Corr



(b) Sharpe



(c) Calmar

Figure 4.22: 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.

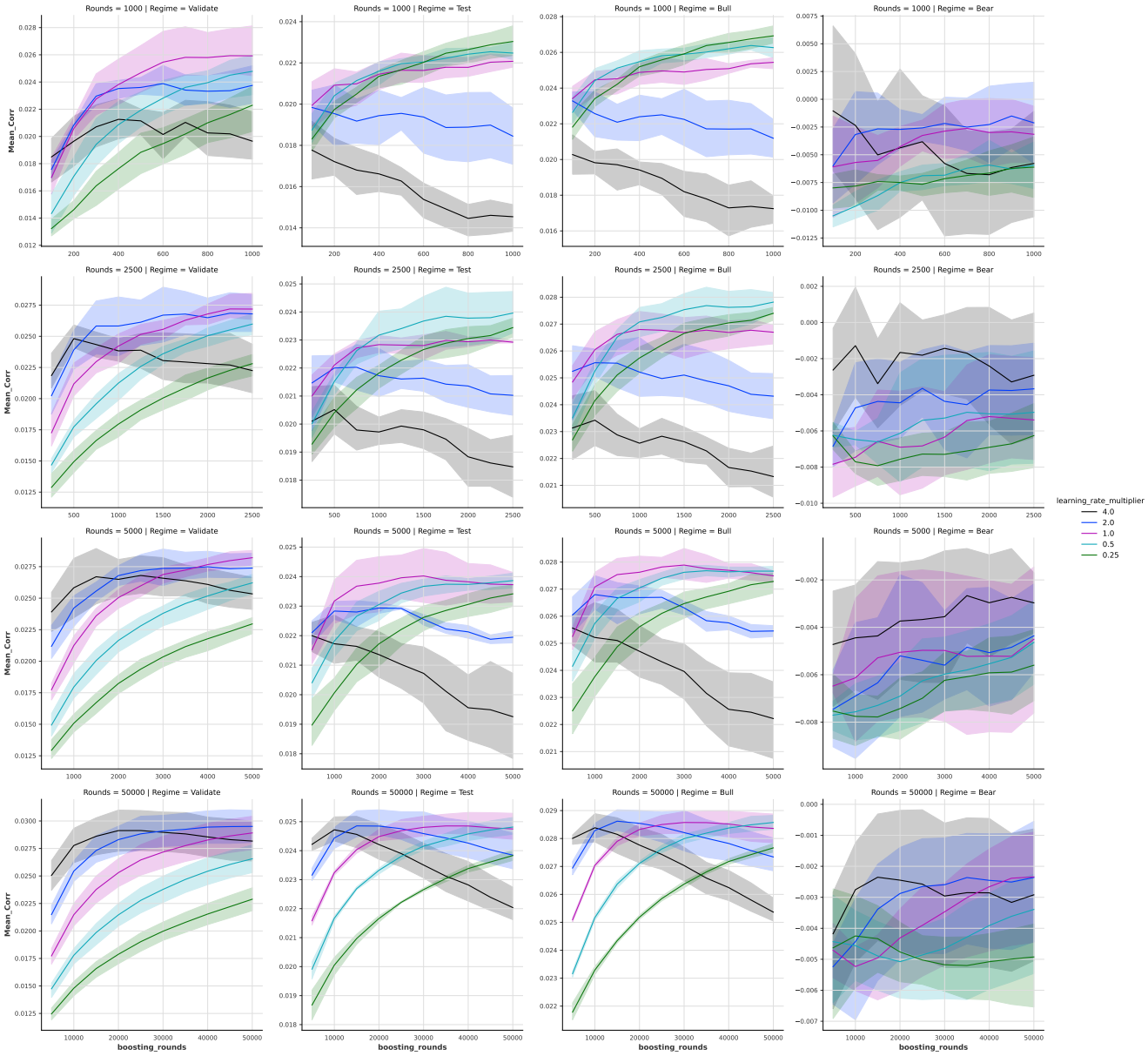


Figure 4.23: Learning curves of XGBoost models with different learning rates for different number of boosting rounds $B = 1000, 2500, 5000, 50000$ for risk metric Mean Corr under different market regimes

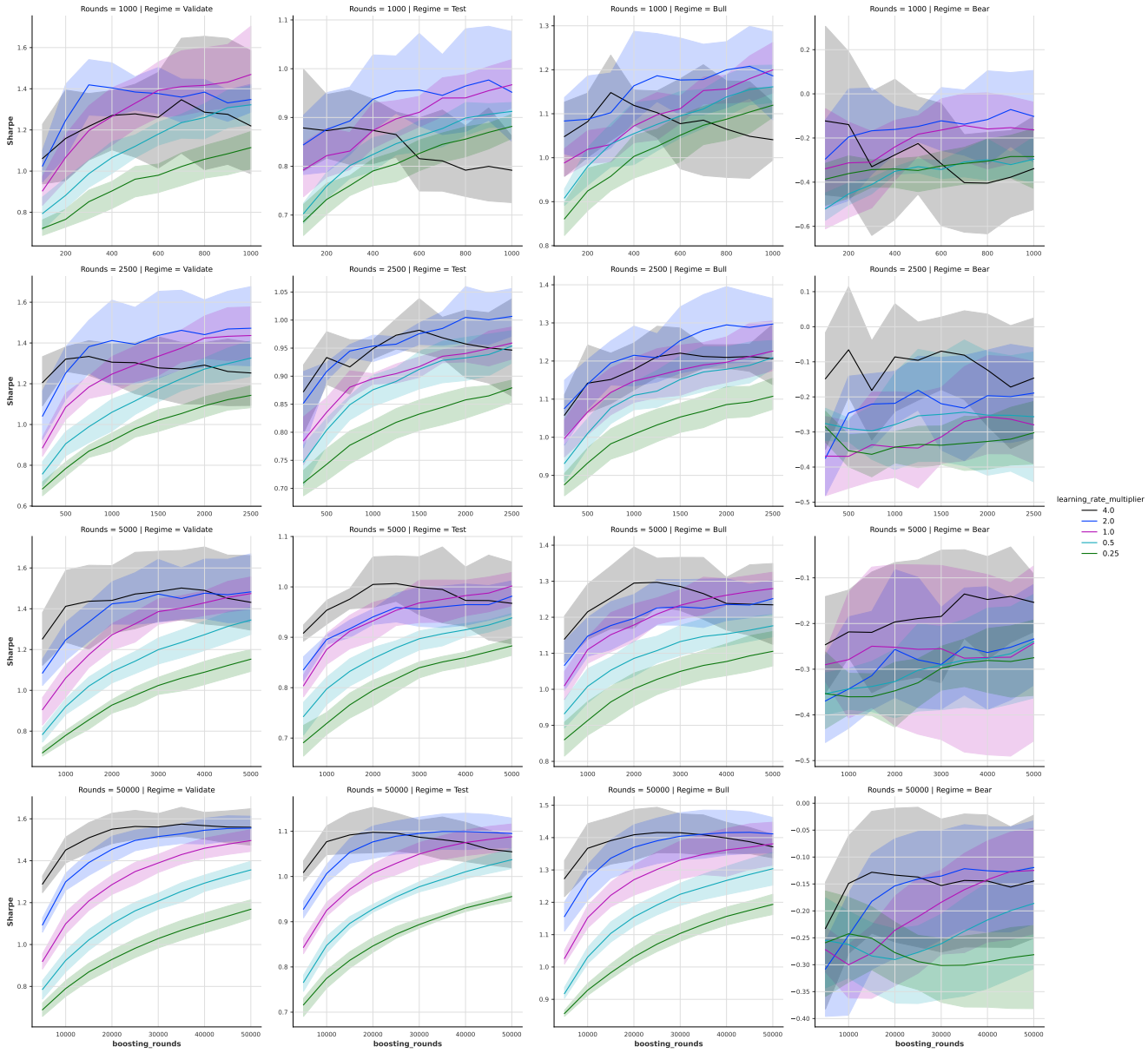


Figure 4.24: Learning curves of XGBoost models with different learning rates for different number of boosting rounds $B = 1000, 2500, 5000, 50000$ for risk metric Sharpe ratio under different market regimes

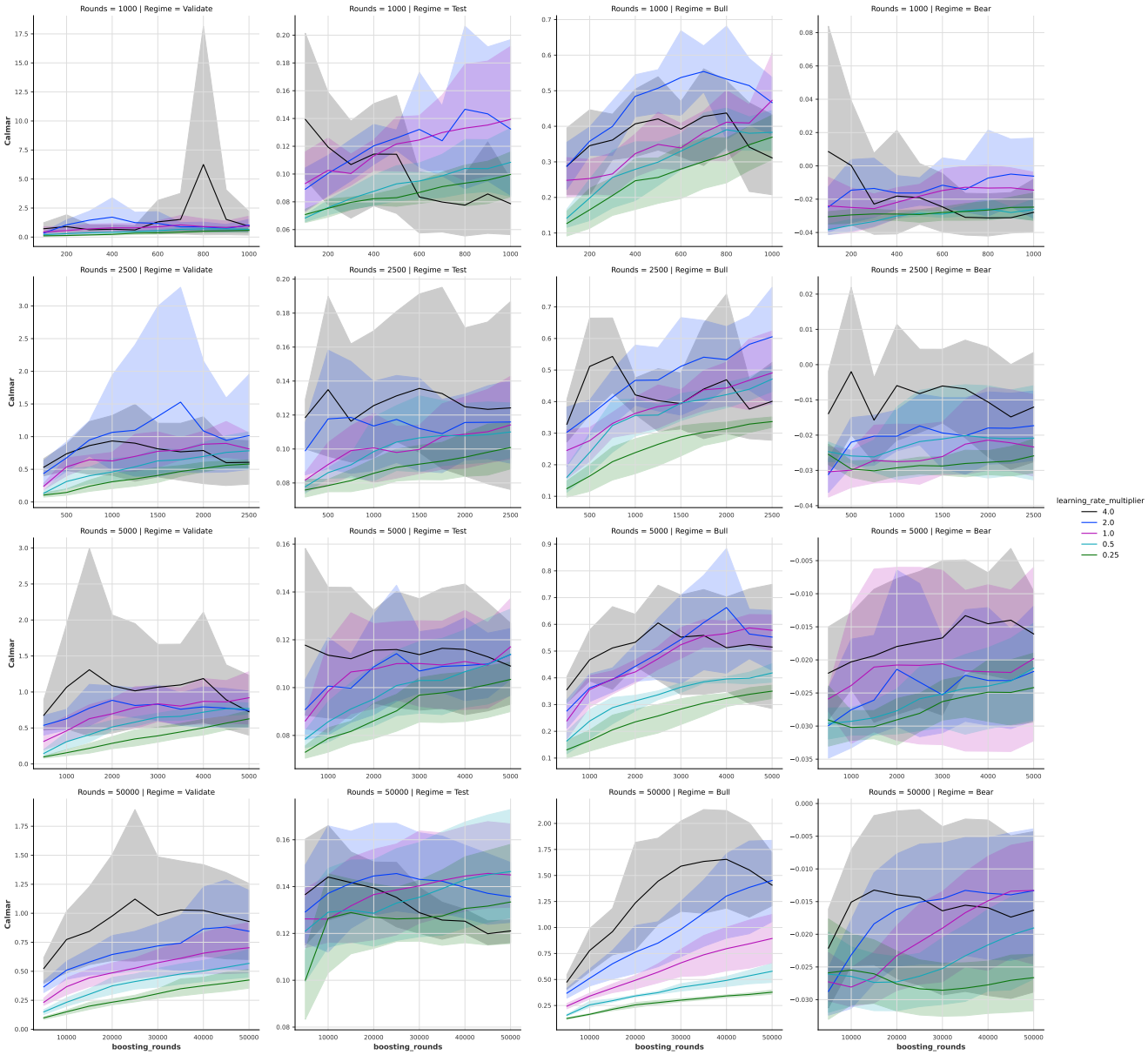


Figure 4.25: Learning curves of XGBoost models with different learning rates for different number of boosting rounds $B = 1000, 2500, 5000, 50000$ for risk metric Calmar ratio under different market regimes

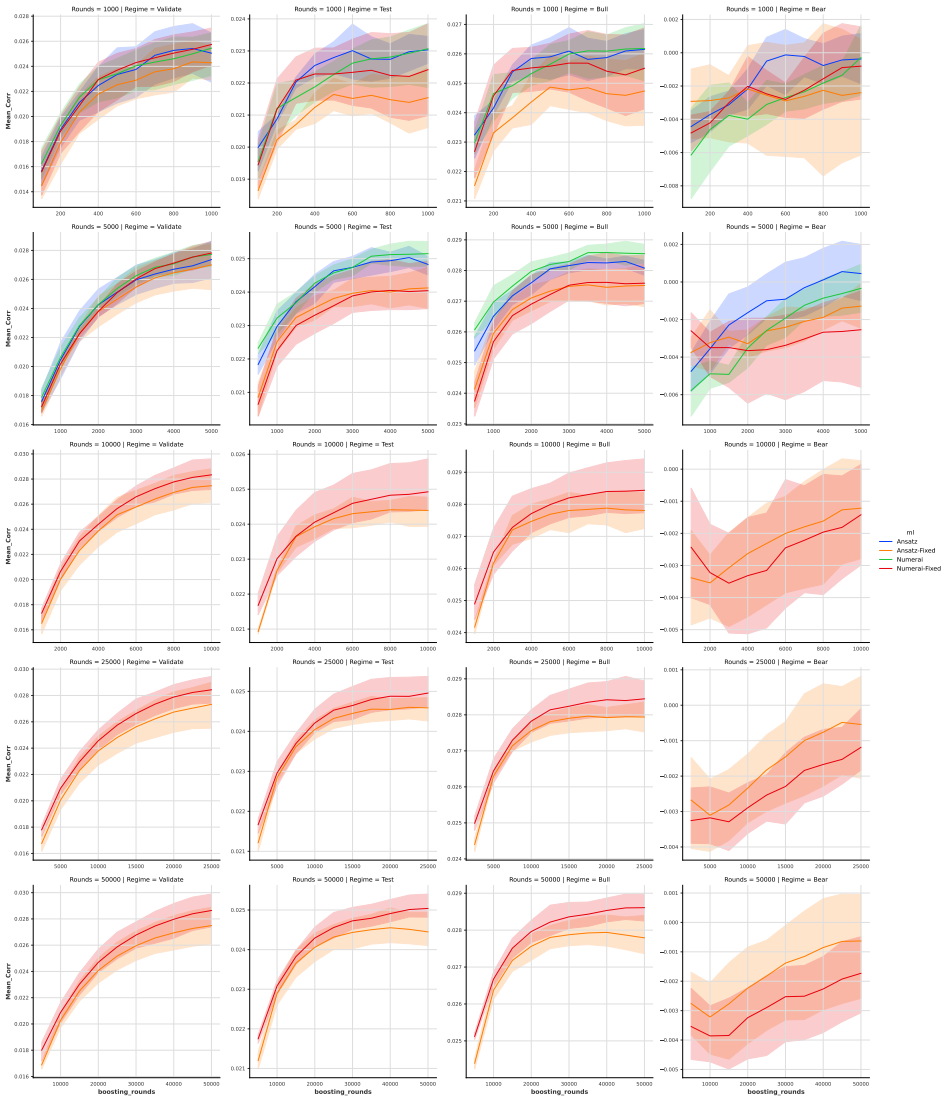


Figure 4.26: Learning curves of benchmark XGBoost models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$ for risk metric Mean Corr under different market regimes

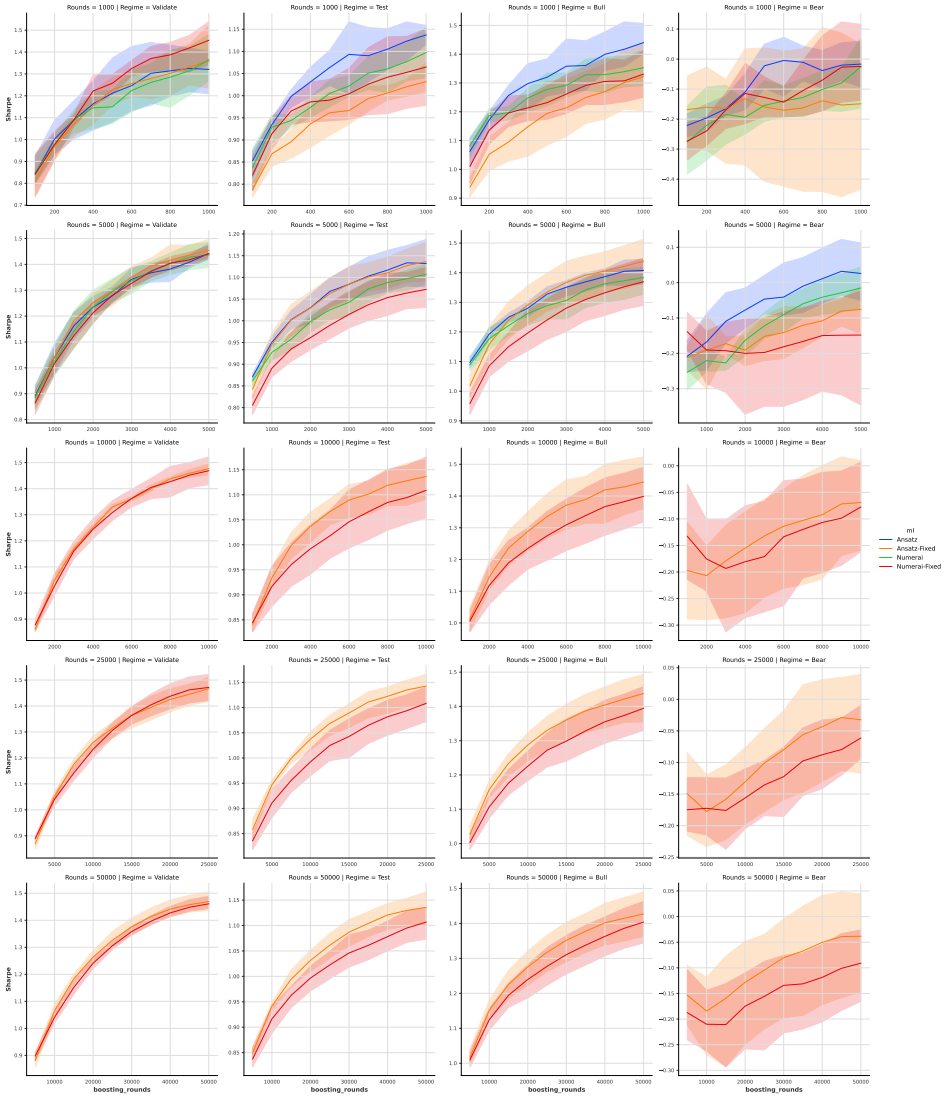


Figure 4.27: Learning curves of benchmark XGBoost models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$ for risk metric Sharpe under different market regimes

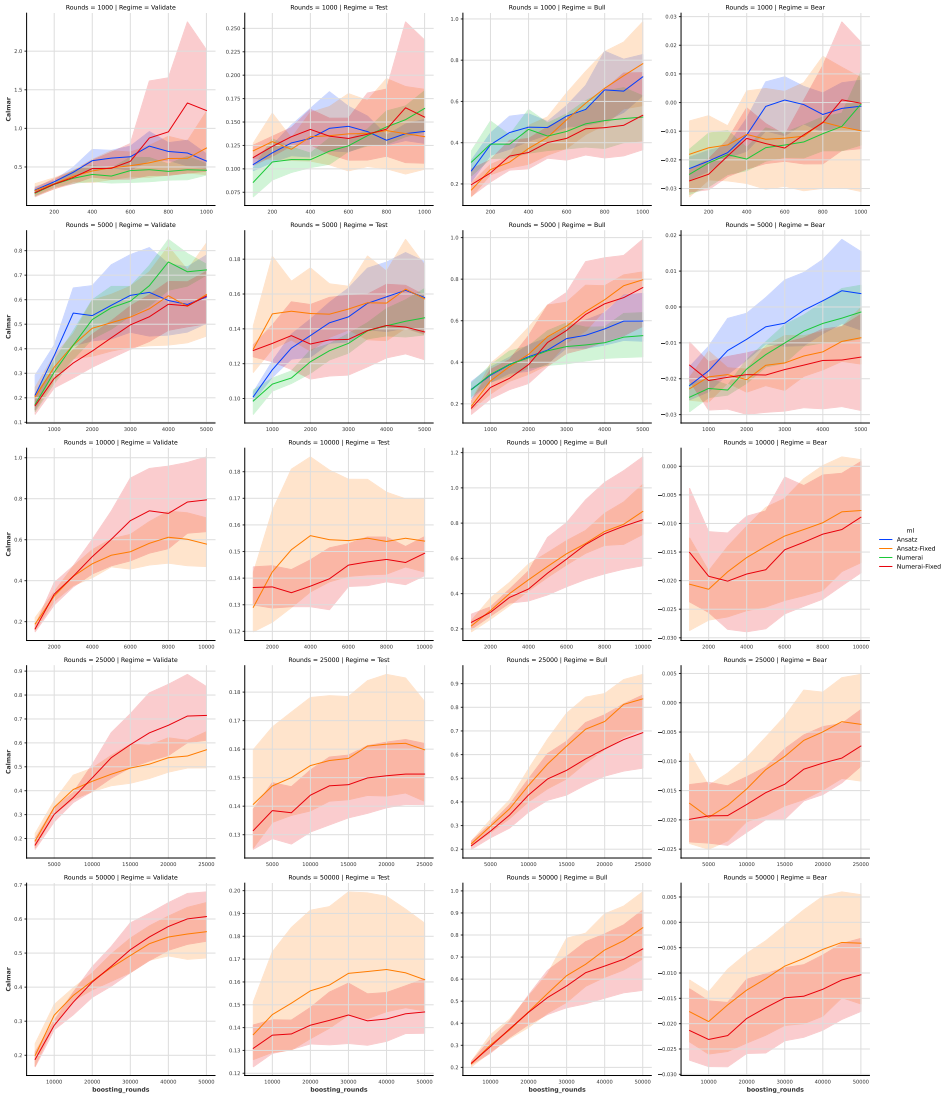
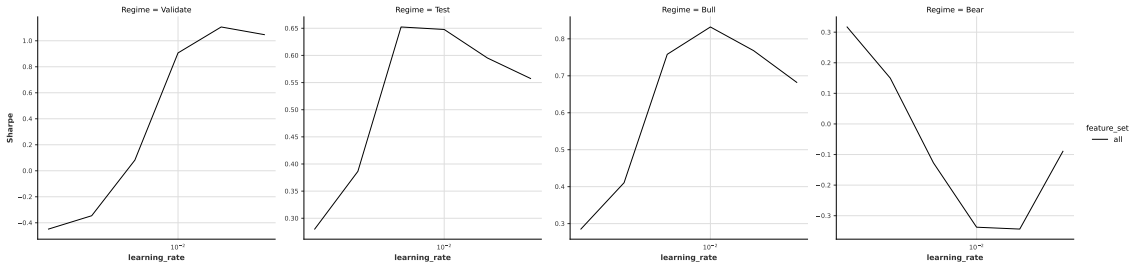
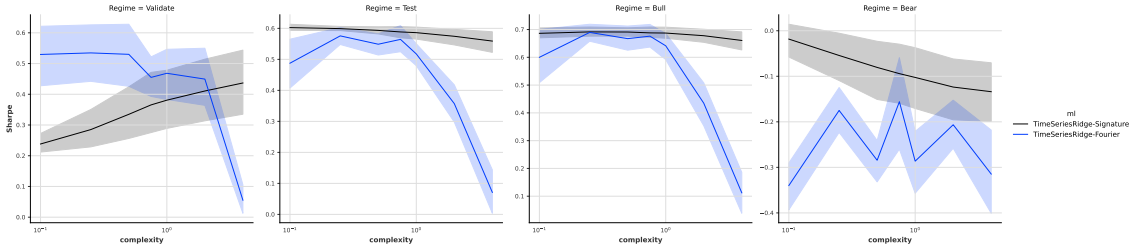


Figure 4.28: Learning curves of benchmark XGBoost models with different number of boosting rounds $B = 1000, 5000, 10000, 25000, 50000$ for risk metric Calmar under different market regimes

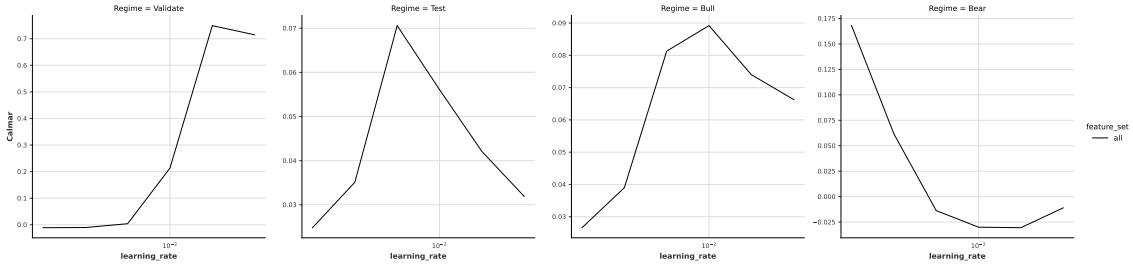


(a) EMA models

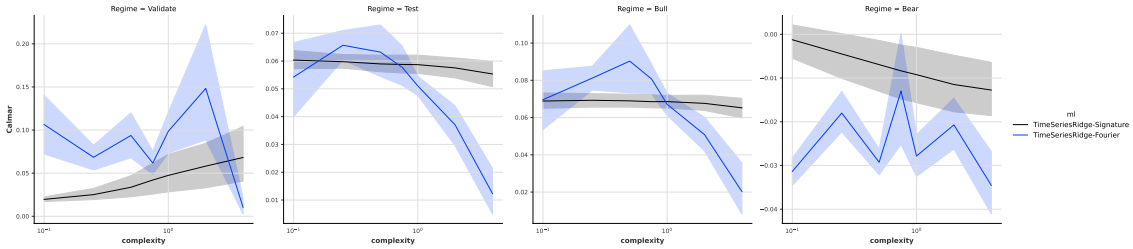


(b) Feature Transform factor-timing models

Figure 4.29: Sharpe ratio of EMA and Feature Transform factor-timing models under different market regimes



(a) EMA models



(b) Feature Transform factor-timing models

Figure 4.30: 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	0.0266	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 4.7: 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.

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 4.8: 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.

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 4.9: 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.

Chapter 5

Conclusion

In this thesis, we developed robust machine-learning methods for studying high-dimensional panel datasets under different applications, including both unsupervised learning tasks of latent factors discovery and clustering and supervised learning tasks of regression-based item ranking. We improved existing methods for dimensionality reduction for batch tabular data under the context of transfer learning. We also built a new framework for incremental learning for temporal tabular datasets under concept drifts.

In chapter 2, we extended Hierarchical Poisson Factorisation (HPF), a commonly used matrix factorisation method for data integration in scRNA-seq datasets. The newly developed method, Integrative Hierarchical Poisson Factorisation (IHPF) can be applied to different data integration scenarios robustly and provide interpretable latent factors, which captures biological signals from the datasets.

The machine learning methods we developed in chapter 2 can be used by researchers to study scRNA-seq datasets and other genomics datasets. Our single-cell methodology directly applies to the expanding body of datasets generated through next-generation single-cell transcriptomics technologies. The method is based on algorithms initially designed for the analysis of text (sparse vectors) and leads to interpretable outcomes, in the terms of gene clusters associated with different conditions. It can also be used to cluster cells for cell type discovery. Hence one of the advantages of the approach is in the interpretability of the results. The second key aspect is that it allows controlling the integration of diverse datasets with a single parameter, for example from different labs or experimental setups.

In chapter 3, we developed simple rule-based and statistical methods, which can be used to adapt different machine learning models towards regime changes in temporal panel datasets. The modularised nature of our methods allows them to be applied towards predictions obtained from any machine learning methods. Different dynamic optimisation techniques, such as dynamic feature projection and dynamic model selection/stacking can be used to improve the robustness of model predictions. Using datasets from Numerai as a case study, we demonstrated the proposed dynamic optimisation techniques can improve the robustness of two important classes of machine learning models, Gradient Boosting Decision Trees (GBDT) and Multi-Layer Perceptron Networks (MLP) over a wide range of parameter settings in model training.

The methodologies developed in chapter 3 can be generalised to deal with temporal datasets presented in tabular form, where observations from a population are collected regularly and features of each observation are pre-processed and standardised.

One then wants to use the observed high-dimensional vectors to predict a ranking or a continuous outcome for the samples. The new observations can be incorporated into the prediction model, but importantly, the properties of the population from which the samples are extracted can be time varying,

making this task non-stationary and thus incremental learning difficult. Our methods resolve some of these issues by evaluating how to project out features that are less predictable and by selecting ensembles of models that are highly resistant to over-fitting of the observed history. This is achieved by producing highly diverse models of limited complexity, where the model diversity provides improvements in the prediction tasks.

There are no domain-specific assumptions made in the methodologies. Indeed, the model can be applied to any standardised temporal panel datasets, where a collection of features is collected over time and a collection of samples is extracted regularly, on which a prediction tasks need to be carried out, for example ranking a set of patients according to some regularly collected features. Our techniques are based on incremental learning and use limited computational resources to update the prediction models so that they can incorporate new information without the need to recompute the models from scratch. This kind of problem is very complicated when the observed data is not stationary and undergoes regime changes, as are typical in human population studies.

In chapter 4, we developed a new deep incremental learning pipeline for temporal tabular datasets with distribution shifts.

We used a wide range of time-series forecast algorithms to build factor timing models for temporal tabular datasets. None of these algorithms performed better than traditional tabular data models, such as GBDT and MLP. We demonstrated that GBDT models outperform deep learning approaches, agreeing with the conclusions from various benchmark studies [14, 15]. More complex neural architectures are shown to have a higher variance in model performance's and prone to overfitting. However, we derive an Ansatz formula which can find the optimal learning rate of a GBDT model for any given number of boosting rounds such that overfitting does not occur. Under this scenario, we shown that model performances of GBDT models with small perturbations in hyperparameters of tree learning converges to the asymptotic behaviour as the number of boosting rounds increases. We demonstrated that large GBDT models with a shallow tree structure are more robust and scalable than MLP models with a much lower variance in model performances. Our deep incremental algorithm combined two widely used techniques, incremental learning and model stacking to dynamically adapt towards concept drift in data. We studied different strategies to create model ensembles in order to improve robustness of predictions. Mixing model with different learning rates and training set sizes can reduce downside risk of models. Using various feature sampling strategies, such as Jackknife feature sampling, can create diverse models for model stacking. Combining models trained with different targets designed with risk management and hedging strategies can further reduce the downside risks of model during unfavourable market regimes. Hedging involves a trade off of some portfolio profits during favourable market regimes for the reduction of volatility and downside risks during periods where the unhedged strategy had poor performances. The overall risk-adjusted return profile of the hedged strategy is improved. Using the variance between models within the ensemble as a measure of uncertainty and confidence in predictions, we created tail risk strategy acting as a good complement to the baseline ensemble strategy based on simple averages during market regime changes. Using ideas from dynamic model selection, we developed a dynamic hedging strategy that can reduce the drawdown of strategies significantly during Bear market with very little trade off to portfolio profits during Bull market. Combining all the above techniques, we created a dynamically hedged deep incremental ensemble model based on XGBoost and Jackknife feature sampling that performed better than the example model provided by Numerai with improved risk profile.

Very recently (Apr 2023), other researchers have begun reporting similar approaches to those we presented in 3 and 4. OFTER, an online pipeline for time series forecasting [191] combines three different approaches, Generalised Regression Neural Network (GRNN), K-Nearest Neighbours (kNN)

and Ordinary Least Squares (OLS) and uses a dynamic switch based on recent performances at each online prediction. Some research [17] also reported standard neural network architectures, such as LeNet5 [192], are susceptible to variation due to random seeds. Model convergences are also highly dependent on the initial weights, a byproduct of random seeds, which results in highly varied training time. These observations agreed with our findings in 4 which demonstrated larger neural networks have more varied performances.

5.1 Informal Summary of Results

Fragile ML vs Anti-Fragile ML

Example: Deep learning is fragile, adversarial examples on noise for image classification

Example: Ensemble Learning models such as GBDT/RF are robust against noise. Using hierarchical modelling to further ensemble XGBoost models can even more robust models

Example: Decision Rules Collections evolved using genetic algorithms are robust and it is very flexible

Degree of freedom are useful when well designed

Example: IHPF add noise ratio parameters, it increases variances and degree of freedom of models. However model performances are improved in all scenarios. scVI, a deep learning based model only work for some scenarios. Both attempt to model batch effects by using extra parameters but IHPF is more robust when started with a small initial guess.

Over-parameterised models works best when there are no regime changes. However under regime changes simple model are better. A barbell approach to combine the simplest and the most complicated models are better

Example: GBDT models are more robust than MLP models under Bear market. In Bull market, both GBDT and MLP models performed similarly.

Example: Dynamic feature projection and model selection beats RL under limited data setting in Paper 2.

Example: Average Ensemble are better than Elastic Net and other selection methods in Paper 3.

Hyper-parameter optimisation is a noisy process and not very replicable

Example: Hyper-parameters for GBDT, training sizes and learning rates under small perturbations do not change performances significantly in paper 2 and 3

Bibliography

[1] J. Gama et al. "A Survey on Concept Drift Adaptation". In: *ACM Comput. Surv.* 46.4 (Mar. 2014). ISSN: 0360-0300. DOI: 10.1145/2523813. URL: https://doi.org/10.1145/2523813.

[2] J. Lu et al. "Learning under Concept Drift: A Review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2019), pp. 2346–2363. DOI: 10.1109/TKDE.2018.2876857.

[3] F. Bayram, B. S. Ahmed, and A. Kassler. "From concept drift to model degradation: An overview on performance-aware drift detectors". In: *Knowledge-Based Systems* 245 (2022), p. 108632. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2022.108632. URL: https://www.sciencedirect.com/science/article/pii/S0950705122002854.

[4] A. Bifet and R. Gavaldà. "Learning from Time-Changing Data with Adaptive Windowing". In: *Proceedings of the 7th SIAM International Conference on Data Mining* 7 (Apr. 2007), pp. 443–448. DOI: 10.1137/1.9781611972771.42.

[5] H. M. Gomes et al. "Adaptive random forests for evolving data stream classification". In: *Machine Learning* 106.9 (Oct. 2017), pp. 1469–1495. ISSN: 1573-0565. DOI: 10.1007/s10994-017-5642-8. URL: https://doi.org/10.1007/s10994-017-5642-8.

[6] P. Li et al. "Learning concept-drifting data streams with random ensemble decision trees". In: *Neurocomputing* 166 (2015), pp. 68–83. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.04.024. URL: https://www.sciencedirect.com/science/article/pii/S0925231215004713.

[7] J. Z. Kolter and M. A. Maloof. "Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts". In: *J. Mach. Learn. Res.* 8 (Dec. 2007), pp. 2755–2790. ISSN: 1532-4435.

[8] X.-C. Yin, K. Huang, and H.-W. Hao. "DE2: Dynamic ensemble of ensembles for learning nonstationary data". In: *Neurocomputing* 165 (2015), pp. 14–22. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2014.06.092. URL: https://www.sciencedirect.com/science/article/pii/S0925231215004270.

[9] L. Pietruczuk et al. "A method for automatic adjustment of ensemble size in stream data mining". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016, pp. 9–15. DOI: 10.1109/IJCNN.2016.7727174.

[10] H. Yang et al. "A Very Fast Decision Tree Algorithm for Real-Time Data Mining of Imperfect Data Streams in a Distributed Wireless Sensor Network". In: *International Journal of Distributed Sensor Networks* 8.12 (2012), p. 863545. DOI: 10.1155/2012/863545. eprint: https://doi.org/10.1155/2012/863545. URL: https://doi.org/10.1155/2012/863545.

[11] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[12] G. Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[13] L. Prokhorenkova et al. "CatBoost: unbiased boosting with categorical features". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf.

[14] L. Grinsztajn, E. Oyallon, and G. Varoquaux. "Why do tree-based models still outperform deep learning on typical tabular data?" In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 507–520.

[15] R. Shwartz-Ziv and A. Armon. *Tabular Data: Deep Learning is Not All You Need*. 2021. DOI: 10.48550/ARXIV.2106.03253. URL: https://arxiv.org/abs/2106.03253.

[16] D. McElfresh et al. *When Do Neural Nets Outperform Boosted Trees on Tabular Data?* 2023. arXiv: 2305.02997 [cs.LG].

[17] O. E. Gundersen et al. *Sources of Irreproducibility in Machine Learning: A Review*. 2023. arXiv: 2204.07610 [cs.LG].

[18] J. Frankle and M. Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rJl-b3RcF7.

[19] S. Ö. Arik and T. Pfister. "TabNet: Attentive Interpretable Tabular Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.8 (May 2021), pp. 6679–6687. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16826.

[20] C. Shorten and T. M. Khoshgoftaar. "A survey on image data augmentation for deep learning". In: *Journal of big data* 6.1 (2019), pp. 1–48.

[21] X. Bouthillier et al. "Accounting for variance in machine learning benchmarks". In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 747–769.

[22] S.-M. Moosavi-Dezfooli et al. "Universal adversarial perturbations". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1765–1773.

[23] A. Kadra et al. "Well-tuned simple nets excel on tabular datasets". In: *Advances in neural information processing systems* 34 (2021), pp. 23928–23941.

[24] S. Latifi, N. Mauro, and D. Jannach. "Session-aware recommendation: A surprising quest for the state-of-the-art". In: *Information Sciences* 573 (2021), pp. 291–315. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2021.05.048. URL: https://www.sciencedirect.com/science/article/pii/S0020025521005089.

[25] N. Reimers and I. Gurevych. "Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 338–348. DOI: 10.18653/v1/D17-1035. URL: https://aclanthology.org/D17-1035.

[26] M. Köppen. "The curse of dimensionality". In: *5th online world conference on soft computing in industrial applications (WSC5)*. Vol. 1. 2000, pp. 4–8.

[27] Numerai. *Numerai Hedge Fund*. (2023, Feb 15). URL: https://numer.ai/data/v4.1.

[28] Numerai. *Numerai Hedge Fund*. (2023, Sep 6). URL: https://numer.ai/data/v4.2.

[29] O. E. Gundersen et al. "On Reporting Robust and Trustworthy Conclusions from Model Comparison Studies Involving Neural Networks and Randomness". In: *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability*. ACM REP '23. Santa Cruz, CA, USA: Association for Computing Machinery, 2023, pp. 37–61. ISBN: 9798400701764. DOI: 10.1145/3589806.3600044. URL: https://doi.org/10.1145/3589806.3600044.

[30] H. V. Pham et al. "Problems and Opportunities in Training Deep Learning Software Systems: An Analysis of Variance". In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ASE '20. Virtual Event, Australia: Association for Computing Machinery, 2021, pp. 771–783. ISBN: 9781450367684. DOI: 10.1145/3324884.3416545. URL: https://doi.org/10.1145/3324884.3416545.

[31] D. Wolpert and W. Macready. "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893.

[32] X. Tang et al. "The single-cell sequencing: new developments and medical applications". In: *Cell & Bioscience* 9.1 (2019), p. 53. ISSN: 2045-3701.

[33] B. Hwang, J. H. Lee, and D. Bang. "Single-cell RNA sequencing technologies and bioinformatics pipelines". In: *Experimental & Molecular Medicine* 50.8 (Aug. 2018), p. 96. ISSN: 2092-6413.

[34] A. Butler et al. "Integrating single-cell transcriptomic data across different conditions, technologies, and species". In: *Nature biotechnology* 36.5 (2018), pp. 411–420.

[35] H. T. N. Tran et al. "A benchmark of batch-effect correction methods for single-cell RNA sequencing data". In: *Genome Biology* 21.1 (Jan. 2020), p. 12.

[36] R. Argelaguet et al. "MOFA+: a statistical framework for comprehensive integration of multi-modal single-cell data". In: *Genome Biology* 21.1 (May 2020), p. 111.

[37] A. Singh et al. "DIABLO: an integrative approach for identifying key molecular drivers from multi-omics assays". In: *Bioinformatics* 35.17 (Jan. 2019), pp. 3055–3062.

[38] R. Lopez et al. "Deep Generative Modeling for Single-cell Transcriptomics". In: *Nature methods* 15.12 (2018), pp. 1053–1058.

[39] B. Hie, B. Bryson, and B. Berger. "Efficient integration of heterogeneous single-cell transcriptomes using Scanorama". In: *Nature Biotechnology* 37.6 (2019), pp. 685–691.

[40] S. Sun et al. "Accuracy, robustness and scalability of dimensionality reduction methods for single-cell RNA-seq analysis". In: *Genome Biology* 20.1 (2019), p. 269.

[41] R. Petegrosso, Z. Li, and R. Kuang. "Machine learning and statistical methods for clustering single-cell RNA-sequencing data". In: *Briefings in Bioinformatics* 21.4 (June 2019), pp. 1209–1223. ISSN: 1477-4054.

[42] G. H. Golub. *Matrix computations*. eng. Fourth edition. Johns Hopkins studies in the mathematical sciences. Baltimore, Maryland: The Johns Hopkins University Press, 2013. ISBN: 1421407949.

[43] D. D. Lee and H. S. Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401 (1999), pp. 788–791.

[44] M. E. Tipping and C. M. Bishop. "Probabilistic Principal Component Analysis". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61.3 (1999), pp. 611–622.

[45] C. M. Bishop. *Pattern recognition and machine learning*. eng. Information science and statistics. New York, NY, USA: Springer, 2006 - 2006. ISBN: 9780387310732.

[46] A. Kucukelbir et al. "Automatic Differentiation Variational Inference". In: *Journal of Machine Learning Research* 18.14 (2017), pp. 1–45.

[47] M. Zhou and L. Carin. "Negative Binomial Process Count and Mixture Modeling". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (Feb. 2015), pp. 307–320. DOI: 10.1109/tpami.2013.211.

[48] H. M. Levitin et al. "De novo gene signature identification from single-cell RNA-seq with hierarchical Poisson factorization". In: *Molecular Systems Biology* 15.2 (2019), e8557.

[49] K. Polański et al. "BBKNN: fast batch alignment of single cell transcriptomes". In: *Bioinformatics* 36.3 (Aug. 2019), pp. 964–965. ISSN: 1367-4803.

[50] Z. Yang and G. Michailidis. "A non-negative matrix factorization method for detecting modules in heterogeneous omics multi-modal data". In: *Bioinformatics (Oxford, England)* 32.1 (2016), pp. 1–8.

[51] P. A. Szabo et al. "Single-cell transcriptomics of human T cells reveals tissue and activation signatures in health and disease". In: *Nature Communications* 10.1 (2019), p. 4706.

[52] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. "Variational Inference: A Review for Statisticians". In: *Journal of the American Statistical Association* 112 (Jan. 2016), pp. 859–877.

[53] G. X. Y. Zheng et al. "Massively parallel digital transcriptional profiling of single cells". In: *Nature Communications* 8.1 (Jan. 2017), p. 14049. ISSN: 2041-1723.

[54] M. Baron et al. "A Single-Cell Transcriptomic Map of the Human and Mouse Pancreas Reveals Inter- and Intra-cell Population Structure". In: *Cell Systems* 3.4 (Oct. 2016), 346–360.e4. ISSN: 2405-4712.

[55] M. J. Muraro et al. "A Single-Cell Transcriptome Atlas of the Human Pancreas". In: *Cell systems* 3.4 (Oct. 2016), 385–394.e3.

[56] N. Lawlor et al. "Single-cell transcriptomes identify human islet cell signatures and reveal cell-type-specific expression changes in type 2 diabetes". In: *Genome research* 27.2 (Feb. 2017), pp. 208–222.

[57] Å. Segerstolpe et al. "Single-Cell Transcriptome Profiling of Human Pancreatic Islets in Health and Type 2 Diabetes". In: *Cell metabolism* 24.4 (Oct. 2016), pp. 593–607.

[58] D. Grün et al. "De Novo Prediction of Stem Cell Identity using Single-Cell Transcriptome Data". In: *Cell stem cell* 19.2 (Aug. 2016), pp. 266–277.

[59] F. A. Wolf, P. Angerer, and F. J. Theis. "SCANPY: large-scale single-cell gene expression data analysis". In: *Genome Biology* 19.1 (Feb. 2018), p. 15. ISSN: 1474-760X.

[60] M. D. Luecken et al. "Benchmarking atlas-level data integration in single-cell genomics". In: *Nature Methods* 19.1 (Jan. 2022), pp. 41–50. ISSN: 1548-7105. DOI: 10.1038/s41592-021-01336-8. URL: https://doi.org/10.1038/s41592-021-01336-8.

[61] X. Yan et al. "CLAIRE: contrastive learning-based batch correction framework for better balance between batch mixing and preservation of cellular heterogeneity". In: *Bioinformatics* 39.3 (Feb. 2023), btad099. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btad099. eprint: https://academic.oup.com/bioinformatics/article-pdf/39/3/btad099/49417417/btad099.pdf. URL: https://doi.org/10.1093/bioinformatics/btad099.

[62] Y. Qian et al. "Geometric graphs from data to aid classification tasks with Graph Convolutional Networks". In: *Patterns* 2.4 (2021), p. 100237. ISSN: 2666-3899.

[63] Z. Liu and M. Barahona. "Graph-based data clustering via multiscale community detection". In: *Applied Network Science* 5.1 (2020), p. 3.

[64] E. Brunner and U. Munzel. "The Nonparametric Behrens-Fisher Problem: Asymptotic Theory and a Small-Sample Approximation". In: *Biometrical Journal* 42.1 (2000), pp. 17–25. DOI: https://doi.org/10.1002/(SICI)1521-4036(200001)42:1<17::AID-BIMJ17>3.0.CO;2-U. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291521-4036%28200001%2942%3A1%3C17%3A%3AAID-BIMJ17%3E3.0.CO%3B2-U. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291521-4036%28200001%2942%3A1%3C17%3A%3AAID-BIMJ17%3E3.0.CO%3B2-U.

[65] L. van der Maaten. "Accelerating t-SNE using Tree-Based Algorithms". In: *Journal of Machine Learning Research* 15 (2014), pp. 3221–3245.

[66] G. Brock et al. "clValid: An R Package for Cluster Validation". In: *Journal of Statistical Software* 25.4 (2008), pp. 1–22. DOI: 10.18637/jss.v025.i04. URL: https://www.jstatsoft.org/index.php/jss/article/view/v025i04.

[67] S. Datta and S. Datta. "Methods for evaluating clustering algorithms for gene expression data using a reference set of functional classes". In: *BMC Bioinformatics* 7.1 (Aug. 2006), p. 397. ISSN: 1471-2105.

[68] G. Yu et al. "GOSemSim: an R package for measuring semantic similarity among GO terms and gene products". In: *Bioinformatics* 26.7 (Feb. 2010), pp. 976–978. ISSN: 1367-4803.

[69] Z. Liu and M. Barahona. "Similarity measure for sparse time course data based on Gaussian processes". In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. Ed. by C. de Campos and M. H. Maathuis. Vol. 161. Proceedings of Machine Learning Research. PMLR, 27–30 Jul 2021, pp. 1332–1341.

[70] C. Xu et al. "Probabilistic harmonization and annotation of single-cell transcriptomics data with deep generative models". In: *Molecular Systems Biology* 17.1 (2021), e9620.

[71] R. Lambiotte, J.-C. Delvenne, and M. Barahona. "Random Walks, Markov Processes and the Multiscale Modular Organization of Complex Networks". In: *IEEE Transactions on Network Science and Engineering* 1.2 (2014), pp. 76–90.

[72] L. L. Hsu and A. C. Culhane. "Impact of Data Preprocessing on Integrative Matrix Factorization of Single Cell Data". In: *Frontiers in Oncology* 10 (2020), p. 973.

[73] M. D. Luecken and F. J. Theis. "Current best practices in single-cell RNA-seq analysis: a tutorial". In: *Molecular Systems Biology* 15.6 (2019), e8746.

[74] L. Peng et al. "Single-cell RNA-seq clustering: datasets, models, and algorithms". In: *RNA Biology* 17.6 (2020), pp. 765–783.

[75] A. Sarkar and M. Stephens. "Separating measurement and expression models clarifies confusion in single-cell RNA sequencing analysis". In: *Nature genetics* 53.6 (2021), pp. 770–777.

[76] G. Durif et al. "Probabilistic count matrix factorization for single cell expression data analysis". In: *Bioinformatics* 35.20 (Mar. 2019). Ed. by I. Birol, pp. 4011–4019. ISSN: 1460-2059.

[77] N. X. Vinh, J. Epps, and J. Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance". In: *Journal of Machine Learning Research* 11.95 (2010), pp. 2837–2854.

[78] L. McInnes, J. Healy, and J. Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. arXiv: 1802.03426 [stat.ML].

[79] L. Haghverdi et al. "Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors". In: *Nature Biotechnology* 36.5 (May 2018), pp. 421–427. ISSN: 1546-1696.

[80] W.-H. Chen et al. "Attribute-Aware Recommender System Based on Collaborative Filtering: Survey and Classification". In: *Frontiers in Big Data* 2 (Jan. 2020). ISSN: 2624-909X.

[81] D. P. Kingma and M. Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends in Machine Learning* 12.4 (2019), pp. 307–392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[82] E. Pierson and C. Yau. "ZIFA: Dimensionality reduction for zero-inflated single-cell gene expression analysis". In: *Genome Biology* 16.1 (Nov. 2015), p. 241.

[83] D. Risso et al. "A general and flexible method for signal extraction from single-cell RNA-seq data". In: *Nature Communications* 9.1 (Jan. 2018), p. 284.

[84] D. D. Lee and H. S. Seung. "Algorithms for Non-Negative Matrix Factorization". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00. Denver, CO: MIT Press, 2000, pp. 535–541.

[85] W. Karush. "Minima of functions of several variables with inequalities as side conditions." PhD thesis. Thesis (S.M.)–University of Chicago, Department of Mathematics, December 1939., 1939.

[86] B. Ghojogh et al. *KKT Conditions, First-Order and Second-Order Optimization, and Distributed Optimization: Tutorial and Survey*. 2021. arXiv: 2110.01858 [math.OC].

[87] L. Li, G. Lebanon, and H. Park. "Fast Bregman Divergence NMF Using Taylor Expansion and Coordinate Descent". In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. Beijing, China: Association for Computing Machinery, 2012, pp. 307–315. ISBN: 9781450314626. DOI: 10.1145/2339530.2339582. URL: https://doi.org/10.1145/2339530.2339582.

[88] C. Févotte and J. Idier. "Algorithms for Nonnegative Matrix Factorization with the beta-Divergence". In: *Neural computation* 23 (June 2011), pp. 2421–2456. DOI: 10.1162/NECO_a_00168.

[89] H. Zhao et al. "Variational autoencoders for sparse and overdispersed discrete data". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 1684–1694.

[90] M. D. Hoffman et al. “Stochastic Variational Inference”. In: *Journal of Machine Learning Research* 14.4 (2013), pp. 1303–1347. URL: http://jmlr.org/papers/v14/hoffman13a.html.

[91] S. Amari. “Natural Gradient Works Efficiently in Learning”. In: *Neural Computation* 10.2 (1998), pp. 251–276.

[92] S. Z. Dadaneh et al. “Bayesian gamma-negative binomial modeling of single-cell RNA sequencing data”. In: *BMC genomics* 21 (2020), pp. 1–10.

[93] J. Arosemena et al. “Stock Price Analysis with Deep-Learning Models”. In: *2021 IEEE Colombian Conference on Applications of Computational Intelligence (ColCACI)*. 2021, pp. 1–6. DOI: 10.1109/ColCACI52978.2021.9469554.

[94] S. Selvin et al. “Stock price prediction using LSTM, RNN and CNN-sliding window model”. In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2017, pp. 1643–1647. DOI: 10.1109/ICACCI.2017.8126078.

[95] K. A. Althelaya, E.-S. M. El-Alfy, and S. Mohammed. “Evaluation of bidirectional LSTM for short- and long-term stock market prediction”. In: *2018 9th International Conference on Information and Communication Systems (ICICS)*. 2018, pp. 151–156. DOI: 10.1109/IACS.2018.8355458.

[96] J. Hullman et al. “The Worst of Both Worlds: A Comparative Analysis of Errors in Learning from Data in Psychology and Machine Learning”. In: *Proceedings of the 2022 AAAI/ACM Conference on AI,Ethics, and Society*. ACM, July 2022. DOI: 10.1145/3514094.3534196. URL: https://doi.org/10.1145%2F3514094.3534196.

[97] S. Kapoor and A. Narayanan. *Leakage and the Reproducibility Crisis in ML-based Science*. 2022. arXiv: 2207.07048 [cs.LG].

[98] E. Rivera-Landos, F. Khomh, and A. Nikanjam. “The Challenge of Reproducible ML: An Empirical Study on The Impact of Bugs”. In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. 2021, pp. 1079–1088. DOI: 10.1109/QRS54544.2021.00116.

[99] S. Samuel, F. Löffler, and B. König-Ries. “Machine Learning Pipelines: Provenance, Reproducibility and FAIR Data Principles”. In: *Provenance and Annotation of Data and Processes*. Ed. by B. Glavic, V. Braganholo, and D. Koop. Cham: Springer International Publishing, 2021, pp. 226–230. ISBN: 978-3-030-80960-7.

[100] J. Bollen, H. Mao, and X. Zeng. “Twitter mood predicts the stock market”. In: *Journal of Computational Science* 2.1 (2011), pp. 1–8. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2010.12.007. URL: https://www.sciencedirect.com/science/article/pii/S187775031100007X.

[101] J. P. N and B. Vasudevan. “Effective Implementation of Neural Network Model with Tune Parameter for Stock Market Predictions”. In: *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*. 2021, pp. 1038–1042. DOI: 10.1109/ICOSEC51865.2021.9591781.

[102] S. Singh and S. Sharma. “Forecasting Stock Price Using Partial Least Squares Regression”. In: *2018 8th International Conference on Cloud Computing, Data Science and Engineering (Confluence)*. 2018, pp. 587–591. DOI: 10.1109/CONFLUENCE.2018.8442645.

[103] P. Moritz et al. *Ray: A Distributed Framework for Emerging AI Applications*. 2017. DOI: 10.48550/ARXIV.1712.05889. URL: https://arxiv.org/abs/1712.05889.

[104] F. Zhou et al. “Cascading logistic regression onto gradient boosted decision trees for forecasting and trading stock indices”. In: *Applied Soft Computing* 84 (2019), p. 105747. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2019.105747. URL: https://www.sciencedirect.com/science/article/pii/S1568494619305289.

[105] Numerai. *Numerai Hedge Fund*. (2022, Apr 12). URL: https://numerai.fund/.

[106] D. B. Percival and A. T. Walden. *Spectral Analysis for Univariate Time Series*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2020. DOI: 10.1017/9781139235723.

[107] B. Lim et al. “Temporal fusion transformers for interpretable multi-horizon time series forecasting”. In: *International Journal of Forecasting* 37.4 (2021), pp. 1748–1764.

[108] S. B. Kotsiantis. “Decision trees: a recent overview”. In: *Artificial Intelligence Review* 39 (2011), pp. 261–283.

[109] Y. Freund and R. E. Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[110] J. H. Friedman. “Greedy function approximation: A gradient boosting machine”. eng. In: *The Annals of statistics* 29.5 (2001), pp. 1189–1232. ISSN: 0090-5364.

[111] R. Korlakai Vinayak and R. Gilad-Bachrach. “DART: Dropouts meet Multiple Additive Regression Trees”. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, Sept. 2015, pp. 489–497. URL: https://proceedings.mlr.press/v38/korlakaivinayak15.html.

[112] V. Borisov et al. “Deep neural networks and tabular data: A survey”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[113] S. Popov, S. Morozov, and A. Babenko. *Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data*. 2019. arXiv: 1909.06312 [cs.LG].

[114] A. Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[115] G. Ke et al. “DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 384–394. ISBN: 9781450362016. DOI: 10.1145/3292500.3330858. URL: https://doi.org/10.1145/3292500.3330858.

[116] W. Song et al. “AutoInt”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, Nov. 2019. DOI: 10.1145/3357384.3357925. URL: https://doi.org/10.11452F3357384.3357925.

[117] T. Akiba et al. “Optuna: A next-generation hyperparameter optimization framework”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[118] J. Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. DOI: 10.1073/pnas.1611835114. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1611835114.

[119] D. E. Kirk. *Optimal control theory : an introduction*. eng. Mineola, N.Y: Dover Publications, 2004 - 1970. ISBN: 0486434842.

[120] R. S. Sutton. *Reinforcement learning : an introduction*. eng. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 9780262352703.

[121] V. Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.

[122] J. Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[123] T. Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html.

[124] J. Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG].

[125] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2014. URL: http://arxiv.org/abs/1312.6114.

[126] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[127] P. Buhlmann and T. Hothorn. “Boosting Algorithms: Regularization, Prediction and Model Fitting”. In: *Statistical Science* 22.4 (Nov. 2007). DOI: 10.1214/07-sts242. URL: https://doi.org/10.1214%2F07-sts242.

[128] G. Ke et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[129] M. Song et al. “In-situ ai: Towards autonomous and incremental deep learning for iot systems”. In: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2018, pp. 92–103.

[130] A. L. Buczak and E. Guven. “A survey of data mining and machine learning methods for cyber security intrusion detection”. In: *IEEE Communications surveys & tutorials* 18.2 (2015), pp. 1153–1176.

[131] E. Belouadah, A. Popescu, and I. Kanellos. “A comprehensive study of class incremental learning algorithms for visual tasks”. In: *Neural Networks* 135 (2021), pp. 38–54.

[132] Y. Wu et al. “Large scale incremental learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 374–382.

[133] G. M. van de Ven, T. Tuytelaars, and A. S. Tolias. "Three types of incremental learning". In: *Nature Machine Intelligence* 4.12 (2022), pp. 1185–1197.

[134] F. Zhu et al. "Class-incremental learning via dual augmentation". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 14306–14318.

[135] Y. Pei et al. "Learning a Condensed Frame for Memory-Efficient Video Class-Incremental Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 31002–31016. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/c8ac22c0d4b263618f2a4f4657948912-Paper-Conference.pdf.

[136] Y. Zou et al. "Margin-Based Few-Shot Class-Incremental Learning with Class-Level Overfitting Mitigation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 27267–27279. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/ae817e85f71ef86d5c9566598e185b89-Paper-Conference.pdf.

[137] J. Lu et al. "Learning under Concept Drift: A Review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2019), pp. 2346–2363. DOI: 10.1109/TKDE.2018.2876857.

[138] F. Bayram, B. S. Ahmed, and A. Kassler. "From concept drift to model degradation: An overview on performance-aware drift detectors". In: *Knowledge-Based Systems* 245 (2022), p. 108632.

[139] K. Arulkumaran et al. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[140] Y. Hong, Y. Jin, and Y. Tang. "Rethinking Individual Global Max in Cooperative Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 32438–32449. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/d112fdd31c830900d1f2e4ccebffb54f-Paper-Conference.pdf.

[141] O. Bastani et al. "Regret Bounds for Risk-Sensitive Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 36259–36269. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/eb4898d622e9a48b5f9713ea1fcff2bf-Paper-Conference.pdf.

[142] K. Zhang et al. "Robust Multi-Agent Reinforcement Learning with Model Uncertainty". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 10571–10583. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/774412967f19ea61d448977ad9749078-Paper.pdf.

[143] Y. Deng et al. "Deep direct reinforcement learning for financial signal representation and trading". In: *IEEE transactions on neural networks and learning systems* 28.3 (2016), pp. 653–664.

[144] D. Acuna, J. Philion, and S. Fidler. "Towards Optimal Strategies for Training Self-Driving Perception Models in Simulation". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 1686–1699. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/0d5bd023a3ee11c7abca5b42a93c4866-Paper.pdf.

[145] J. Kober, J. A. Bagnell, and J. Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[146] K. Arndt et al. "Meta reinforcement learning for sim-to-real domain adaptation". In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2020, pp. 2725–2731.

[147] I. Higgins et al. “Darla: Improving zero-shot transfer in reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1480–1490.

[148] X. Ma, K. Driggs-Campbell, and M. J. Kochenderfer. “Improved robustness and safety for autonomous vehicle control with adversarial reinforcement learning”. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 1665–1671.

[149] A. Mott et al. “Towards interpretable reinforcement learning using attention augmented agents”. In: *Advances in neural information processing systems* 32 (2019).

[150] A. I. Naimi and L. B. Balzer. “Stacked generalization: an introduction to super learning”. In: *European journal of epidemiology* 33 (2018), pp. 459–464.

[151] T. Wong and M. Barahona. *Online learning techniques for prediction of temporal tabular datasets with regime changes*. 2023. arXiv: 2301.00790 [q-fin.CP].

[152] C. Hoffstein, N. Faber, and S. Braun. “Rebalance timing luck: the (dumb) luck of smart beta”. In: *SSRN 3673910* (2020).

[153] G. Hinton. *The Forward-Forward Algorithm: Some Preliminary Investigations*. 2022. DOI: 10.48550/ARXIV.2212.13345. URL: https://arxiv.org/abs/2212.13345.

[154] A. Didisheim, B. Kelly, and S. Malamud. *Deep Regression Ensembles*. 2022. arXiv: 2203.05417 [stat.ML].

[155] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[156] S. Elsayed et al. *Do We Really Need Deep Learning Models for Time Series Forecasting?* 2021. arXiv: 2101.02118 [cs.LG].

[157] T. J. Lyons. “Differential Equations Driven by Rough Paths : Ecole d’Eté de Probabilités de Saint-Flour XXXIV-2004”. eng. In: École d’Été de Probabilités de Saint-Flour, 1908. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. ISBN: 1-280-85347-6.

[158] I. Chevyrev and A. Kormilitzin. *A Primer on the Signature Method in Machine Learning*. 2016. arXiv: 1603.03788 [stat.ML].

[159] T. Lyons and A. D. McLeod. *Signature Methods in Machine Learning*. 2023. arXiv: 2206.14674 [stat.ML].

[160] B. T. Kelly, S. Malamud, and K. Zhou. “The Virtue of Complexity Everywhere”. In: *SSRN* (2022).

[161] D. J. Sutherland and J. Schneider. “On the Error of Random Fourier Features”. In: *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*. UAI’15. Amsterdam, Netherlands: AUAI Press, 2015, pp. 862–871. ISBN: 9780996643108.

[162] V. Haddad, S. Kozak, and S. Santosh. “Factor timing”. In: *The Review of Financial Studies* 33.5 (2020), pp. 1980–2018.

[163] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[164] A. M. Schäfer and H. G. Zimmermann. “Recurrent neural networks are universal approximators”. In: *Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I 16*. Springer. 2006, pp. 632–640.

[165] B. Lakshminarayanan, A. Pritzel, and C. Blundell. “Simple and scalable predictive uncertainty estimation using deep ensembles”. In: *Advances in neural information processing systems* 30 (2017).

[166] F. Wenzel et al. "Hyperparameter Ensembles for Robustness and Uncertainty Quantification". In: *Neural Information Processing Systems (NeurIPS)*. 2020. URL: https://papers.nips.cc/paper/2020/hash/481fbfa59da2581098e841b7afc122f1-Abstract.html.

[167] S. Zaidi et al. "Neural ensemble search for uncertainty estimation and dataset shift". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 7898–7911.

[168] Numerai. *Numerai Hedge Fund*. (2023, Apr 19). URL: https://docs.numer.ai/tournament/correlation-corr.

[169] Numerai. *Numerai Hedge Fund*. (2023, Aug 29). URL: https://numerai.fund/.

[170] D. H. Bailey and M. L. de Prado. "Stop-outs under serial correlation and the triple penance rule". In: *JOURNAL OF RISK* 18.2 (2015), pp. 61–93. ISSN: 1465-1211. DOI: 10.21314/JOR.2015.317.

[171] B. T. Kelly and S. Malamud. "The virtue of complexity in machine learning portfolios". In: *SSRN Electronic Journal* (2021). DOI: 10.2139/ssrn.3984925.

[172] L. Zhao, S. Kong, and Y. Shen. *DoubleAdapt: A Meta-learning Approach to Incremental Learning for Stock Trend Forecasting*. 2023. arXiv: 2306.09862 [q-fin.ST].

[173] T. Hastie et al. "Surprises in high-dimensional ridgeless least squares interpolation". In: *Annals of statistics* 50.2 (2022), p. 949.

[174] P. G. Thoppil et al. "Ensemble forecasting greatly expands the prediction horizon for ocean mesoscale variability". In: *Communications Earth & Environment* 2.1 (May 2021), p. 89. ISSN: 2662-4435. DOI: 10.1038/s43247-021-00151-5. URL: https://doi.org/10.1038/s43247-021-00151-5.

[175] M. Pezeshki et al. "Multi-scale feature learning dynamics: Insights for double descent". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 17669–17690.

[176] M. Geiger et al. "Jamming transition as a paradigm to understand the loss landscape of deep neural networks". In: *Physical Review E* 100.1 (2019), p. 012115.

[177] B. Efron and C. Stein. "The Jackknife Estimate of Variance". In: *The Annals of Statistics* 9.3 (1981), pp. 586–596. DOI: 10.1214/aos/1176345462. URL: https://doi.org/10.1214/aos/1176345462.

[178] T. G. Bali et al. *Machine Forecast Disagreement*. Tech. rep. National Bureau of Economic Research, 2023.

[179] G. A. Carpenter and S. Grossberg. "Normal and amnesic learning, recognition and memory by a neural model of cortico-hippocampal interactions". In: *Trends in neurosciences* 16.4 (1993), pp. 131–137.

[180] K. Wang et al. "Elastic gradient boosting decision tree with adaptive iterations for concept drift adaptation". In: *Neurocomputing* 491 (2022), pp. 288–304. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.03.038. URL: https://www.sciencedirect.com/science/article/pii/S0925231222003320.

[181] A. Liu, J. Lu, and G. Zhang. "Diverse instance-weighting ensemble based on region drift disagreement for concept drift adaptation". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 293–307.

[182] P. Kidger and T. Lyons. "Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU". In: *International Conference on Learning Representations 2021* (2021).

[183] M. Lemercier et al. “Distribution Regression for Sequential Data”. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 13–15 Apr 2021, pp. 3754–3762. URL: https://proceedings.mlr.press/v130/lemercier21a.html.

[184] J. Morrill et al. *A Generalised Signature Method for Multivariate Time Series Feature Extraction*. 2021. arXiv: 2006.00873 [cs.LG].

[185] W. Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: https://github.com/Lightning-AI/lightning.

[186] L. Zimmer, M. Lindauer, and F. Hutter. “Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL”. In: (2021), pp. 3079–3090. DOI: 10.1109/TPAMI.2021.3067763.

[187] Numerai. *Numerai Hedge Fund*. (2023, Aug 29). URL: https://forum.numer.ai/t/super-massive-lgbm-grid-search/6463.

[188] S. Maddock et al. “Federated Boosted Decision Trees with Differential Privacy”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 2249–2263. ISBN: 9781450394505. DOI: 10.1145/3548606.3560687. URL: https://doi.org/10.1145/3548606.3560687.

[189] A. I. Adler and A. Painsky. “Feature Importance in Gradient Boosting Trees with Cross-Validation Feature Selection”. In: *Entropy* 24.5 (2022). ISSN: 1099-4300. DOI: 10.3390/e24050687. URL: https://www.mdpi.com/1099-4300/24/5/687.

[190] N. TeresaHuang, D. W. Hogg, and S. Villar. “Dimensionality Reduction, Regularization, and Generalization in Overparameterized Regressions”. In: *SIAM Journal on Mathematics of Data Science* 4.1 (2022), pp. 126–152. DOI: 10.1137/20M1387821.

[191] N. Michael, M. Cucuringu, and S. Howison. *OFTER: An Online Pipeline for Time Series Forecasting*. 2023. arXiv: 2304.03877 [stat.ML].

[192] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.