1-31-2023

Utilizing Deep Learning for Enhancing Performance on Encrypted Stock Market Data

Surajsinh Prakashchandra Parmar
University of Windsor

Utilizing Deep Learning For Enhancing Performance On Encrypted Stock Market Data

By

Surajsinh Prakashchandra Parmar

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2023

Utilizing Deep Learning For Enhancing Performance On Encrypted Stock Market Data

by

Surajsinh Prakashchandra Parmar

APPROVED BY:

G. Pandher
Odette School of Business

A. Ngom
School of Computer Science

I. Ahmad, Advisor
School of Computer Science

January 25, 2023

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Hedge funds seek to achieve higher returns on investment in the stock market. Typically, they purchase high-quality data at a high price and employ hundreds of individuals working in isolated teams, leading to duplication of research efforts due to secrecy. As such, the data cannot be shared publicly without risking their competitive edge. Numerai is a hedge fund has developed a method for encrypting high-quality stock market data without compromising its predictive power, allowing public sharing of such data in a weekly data science tournament. This method enables anyone to access the data, generate predictions, and submit them for consideration. Submitted predictions can influence Numerai's allocation of capital in the global stock market. The provided time-series data is cleaned and regularized, which comprises millions of samples and 1191 features that has evolved since their inception. The task in the tournament is to predict the probability of a given sample yielding positive returns. The non-stationary nature of the features presents a significant challenge to the participants. Additionally, participants aim to maintain stable predictions over time. Using this data as a supervised regression learning problem, we focused on improving the Sharpe ratio of correlation scores over time. Tree-based models have demonstrated effectiveness in such tasks, while neural networks have shown potential in computer vision and natural language processing. In this thesis, we investigated the incorporation of deep learning methods to improve results through unsupervised and supervised learning. Our findings indicated an improvement in Sharpe ratio over the provided baseline model. We also examined the potential for generating synthetic data using CTGAN.

ACKNOWLEDGEMENTS

I am deeply grateful to Dr. Imran Ahmad for his invaluable feedback and guidance throughout my work. His expertise and experience in the field have been instrumental in the success of my project. I would also like to thank Dr. Gurupadesh Pandher and Dr. Alioune Ngom for graciously serving on my committee at short notice and for their ongoing support and assistance.

I would like to express my sincere gratitude to the Numerai community for their support and assistance throughout my work. The insights, feedback, and expertise shared by members of the community have been invaluable, and I am grateful for the opportunity to have been a part of this exceptional group of individuals.

I am also indebted to my parents, Prakashchandra and Bhartiben Parmar, as well as my grandparents, Dolatsinh and Urmilaben Rathod, and my brother, Karan, for their love and support throughout my academic career. Additionally, I would like to thank my friends for making my university experience unforgettable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Overview

Public equity markets allow anyone to participate in capital investment. Entities, such as hedge funds, utilize all kinds of data and information to gain a marginal edge in the market. Their approaches include looking into historical prices of equities, buying high-quality derivatives' data, using automated bots for high-frequency trading, and using satellite images in store parking yards to forecast sales to find the alpha [1] which is the excess returns compared to a benchmark.

Some investors see using the stock market to attain higher returns on investments as gambling, while others see it as an opportunity to invest with their due diligence and a hypothesis. Investors have conducted different types of research, such as fundamental analysis (FA) and technical analysis (TA). FA involves looking at a company's balance sheet, earnings, and management. Alternately, TA uses candlestick patterns, Open, High, Low, Close, Volume (OHLCV) data, and derivatives data. It utilizes drawing patterns on charts and looking at the technical indicators, such as simple moving average (SMA) and relative strength index (RSI), to build a hypothesis about the company or market.

With the growth of technology, the number of people investing has also increased, making stock-market investment more accessible and common. This change allowed for the automation of trades. Thus, one can automate the trades if certain conditions are met. For example, investors may buy $100 worth of stock if a promising stock price

falls below 10% in a day. Based on this, institutional investors use high-frequency-trading (HFT) backed by smart analysis and predictive technologies. Due to its popularity and perceived effectiveness, HFT accounted for 52% of trading volume in May 2019.

1.2 Research Motivation

The hedge funds use many techniques to find alpha for their investment strategies. They employ talent from different backgrounds such as Physics, Math, Computer Science, etc. These analysts often works in silos,leading to redundancy in research and unimpressive performance. Furthermore, the data cannot be shared publicly as a strategy would loses its edge if everyone is using it in the market. So, secrecy plays a critical role in achieving success.

Numerai Inc. (numer.ai) is a hedge fund that instrumented a way to encrypt the stock market data without losing its predictive structure. This allows them to share their high-quality data publicly without worrying of their secret data being used by anyone else. Numerai make this data publicly available so data scientists across the world can load it, do modelling, and generate predictions. These predictions are then collected by Numerai to make an ensemble model called "The Meta-Model". This ensemble model of predictions is then used to make trades in the global equity market. The meta model is a stake weighted ensemble of all predictions submitted on time. Stake refers to the confidence the data scientists have in their respective model. The staking mechanism allows data scientists (users) to stake Numeraire cryptocurrency on their predictions. The stake is either rewarded or destroyed based on the performance of individual predictions.

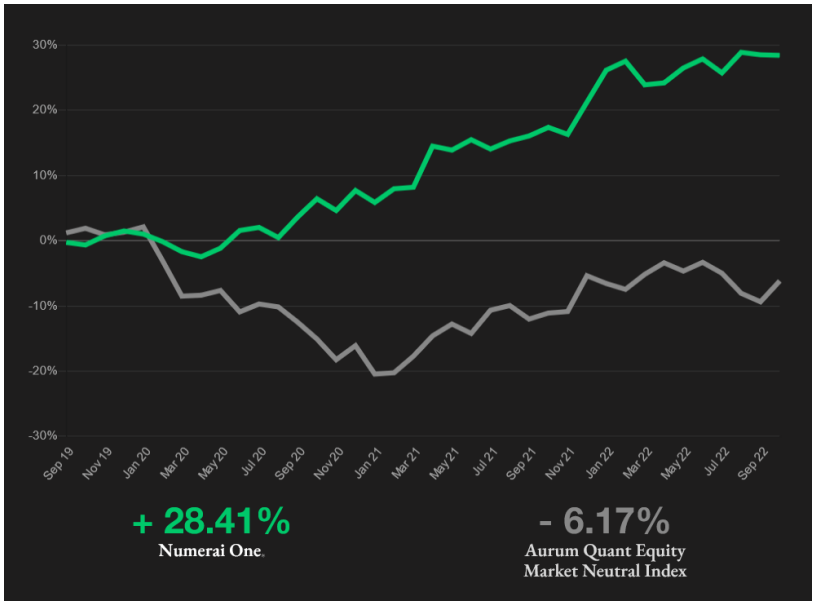


Fig. 1.2.1: Numerai's market-neutral fund performance. [2]

The idea is that an ensemble of predictions from thousands of data scientists should outperform traditional hedge funds. Fig. 1.2.1 shows the effectiveness of this approach of crowd sourcing predictions in the market.

The participants or the data scientists do not need to share their models or code in order to participate in this tournament. They only need to generate predictions using the updated weekly data and submit them. However, if everyone submits the same predictions, then the ensemble would not be diverse enough. Thus, the incentives are to be good and unique than others at the same time. This data science tournament is a weekly tournament. Each week, a new round starts with new data to generate and submit predictions upon.

In this thesis, we used the data provided by Numerai, to achieve the primary goal of combining different set of models using Deep Learning that can add valuable information to the provided example model. Observing the effect of synthetic data is also briefly discussed.

1.3 Research Contribution

Tree based models have been performing well on tabular data in supervised setup. We used deep learning to extract meaningful information from the inputs. Adding this to tree based model helps making the model generalize well and achieve a higher Sharpe ratio. This means, having higher returns per risk.

The main contributions of this thesis are,

- Using de-noising auto-encoder to generate robust latent representations
- Constraining the latent representations to be orthogonal
- Training tree-based models on these latent representations to generate diverse set of predictions
- Thus, a robust framework to combine neural networks and tree-based models.

1.4 Thesis Organization

Chapter 1 outlines the overview, research motive and contributions.

Chapter 2 describes the literature review of how machine learning is mostly applied on stock market data, machine learning on tabular data along with some deep learning ideas related to this thesis.

Chapter 3 explains the dataset, it's previous versions and characteristics.

Chapter 4 discusses machine learning methodologies used in the research and talks about the performance measures used to evaluate the models.

Chapter 5 talks about the proposed methodology with cross-validation strategy and custom loss function.

Chapter 6 states the experimental setup, and results from multiple experiments.

Chapter 7 concludes the research description and provides future research directions.

CHAPTER 2

Literature Review

This chapter discusses some of the significant publications that directed our work. We first discuss about how Machine Learning is applied on stock market data mostly with tree-based models. Then we discuss how diverse machine learning approaches are applied on tabular data as stock market data, at its core is tabular. Followed by related deep learning approaches. We also discuss some of the approaches used on Numerai's previous datasets.

2.1 Machine Learning on stock market data

The use of machine learning to precisely predict the movements of a single stock or a group of stocks has been the subject of numerous studies. To predict various timescales of 1, 2, 5, 10, 15, 20, and 30 days, Nabipour et al. [3] studied and evaluated various tree-based and deep learning algorithms. On the Tehran Stock Exchange, four market segments were chosen: diversified financials, petroleum, non-metallic minerals, and basic metals. Technical indicators were used to generate the features. The daily open, high, low, close, and volume values of these tickers were used to calculate 10 technical indicators, which were then used as features.

In their experiments for tree-based models, Bagging, Random Forest, AdaBoost, Gradient Boosting, and XGBoost were all used. Each one is tuned according to a range of potential parameter values for the number of trees, maximum depth, and learning rate. For deep learning models, Artificial Neural Networks (ANN), Recurrent

Neural Networks (RNN), and Long Short-Term Memory (LSTM) were utilized, each tuned with a variety of potential parameter values as well. The 2600-day dataset was divided into training and testing data at random.

To evaluate their experiments Nabipour et al. [3] used Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), Relative Root Mean squared error, and Mean Squared Error (MSE) as evaluation metrics to compare the performance of various models over time. The decision tree performed poorly due to its non-ensemble nature, but it is the quickest to train. Other tree-based models were outperformed by the AdaBoost regressor. Among the neural network models, LSTM performed the best, while ANN performed the worst. Overall, LSTM performed best, but it has the disadvantage of having the longest runtime due to its sequential processing nature. The authors were impressed by both the tree and neural models and suggested that they be tested on other stock markets in the future.

Basak et al. [4] examined the performance of tree-based models on the stock market. Technical indicators were used as features in their machine learning models in order to accurately predict stock price movement over medium to long term time frames. Price data till 3rd February 2017 for a diverse set of companies was used. Relative Strength Index, Stochastic Oscillator, Williams percentage range, Moving Average Convergence Divergence (MACD), Price Rate of change (PRoC), and On-Balance Volume (OBV) were used as features with a brief description of their reasoning behind selecting specific parameters for these technical indicators. They generated binary targets for multiple windows of 3, 5, 10, 15, 30, 60, and 90 days based on the returns over multiple time frames.

Random forests and XGBoost implementation of Gradient boosted decision trees were compared in their experiments. RoC-AUC was used as the primary metric for evaluation and compared the accuracy of other published methods in a comparative study. The Random Forest model outperformed the XGBoost model in most time-frames. It achieved 78% accuracy compared to 56% and 67% accuracies in research done on Korean and Taiwanese markets respectively. Their reason for random forest outperforming the gradient boosting tree is the extensive use of voting to build forests

in random forests which results in decision trees that are significantly de-correlated. As it carefully investigates the feature space to improve decisions on which class to finalize as the predicted outcome, this course of action is quite ideal for the stock data and associated classification.

Lopez De Prado in the book "Advances in Financial Machine Learning," [5] suggests that K-fold cross-validation fails in finance because the observations aren't necessarily Independent and Identically Distributed (IID). It is possible to have serially correlated features, which means that two consecutive samples may have very similar information and, thus, overlap. Labels can be inherently correlated because they are associated with overlapping data points. Putting them in two separate splits may improve the model's out-of-sample predictions, but it's just over-fitting on very similar in-sample data. As a result of this label leakage, sample features become irrelevant, and researchers may obtain incorrect results.

"Purging" and "Embargoing" are proposed solutions by Lopez De Prado to handle this problem. We can have data where the labels of training data overlap with the labels of test set samples. Purging is the process of removing these overlapping samples from the training set. In addition, features can have serial correlation, in which case, we should remove a set of samples from the training set that immediately follows a sample in the test set. Some common ML failures in finance were discussed as well.

2.2 Machine Learning on Tabular data

Arik et al. [6] proposed a new deep learning-based architecture for tabular data. Tabular data may contain different types of data. It is an encoder-decoder model utilizing attention mechanism. TabNet [6] can be used for both supervised learning and unsupervised pre-training. It also emphasizes on interpretability of the model.

It uses Transformer encoder-decoder built on attention mechanism [7]. The encoder processes the input using a feature transformer, an attentive transformer and masking. The learnable mask helps the attention mechanism to focus on useful fea-

tures on each step. This mask also seconds as global feature importance tool for model interpretation. It is often practiced selecting a subset of features. The authors argued that the TabNet can perform well without explicit feature selection. It can utilize all available features to make the most learning from the data. The feature transformer uses a 4-layer network such that 2 are shared and 2 are independent for a particular decision step. It makes extensive use of Batch Normalization and Gated Linear Unit.

The unsupervised pre-training uses explicit masking of input cells for encoder and allowing the model to learn the relationship using adjacent cells at decoder stage in order to re-construct the original input. The authors tested this architecture on multiple real world and synthetic datasets and compared it with many tree-based tabular data learning methods. The authors concluded that the TabNet was able to outperform tree-based models

Shwartz-Ziv et al. [8] commented that some studies have recently shown that deep neural networks such as TabNet [6], NODE [9], DNF-Net [10], and 1D-CNN [11] could perform better than tree-based models. There is no standard benchmark in tabular data, so each method used different sets of datasets unlike Computer vision having "ImageNet" [12]. The authors compared multiple tree-based methods like CatBoost [13] and XGBoost [14] against a variety of proposed deep neural networks with the goal of a fair comparison. It was found that the deep learning models were outperformed by XGBoost without much tuning. However, an ensemble of XGBoost with neural networks outperformed both model when compared to individual models.

The dataset used in this thesis is tabular. We found some research on very early versions of Numerai's dataset. Thakur et al. [15] used CTree models on Numerai dataset with following techniques in the model: multivariate regression, ordinal regression, censored regression, and recursive binary partitioning. Singh et al. [16] experimented with Partial Least Square Regression in similar manner. They found that a neural network would achieve similar performance but would take longer to train and argued that tuning a neural network could propose a big challenge. Artificial neural networks were used by Jeipratha et al. [17] on an older version of the

Numerai dataset when metrics and targets were different than the latest one with only 21 features. They used grid search cross validation to tune the parameters on the Binary cross entropy metric. They utilized batch normalization and dropouts for regularization. This research focused more on tuning the neural network parameters using 80/20-train/test split rather than comparing Neural networks to other tree-based methods and suggested that other methods like Support Vector Machine (SVM) and Random Forests can also be used.

Singh et al. [18] used another version of the Numerai dataset which had 310 features and 1 target named "target_kazutsugi" with about 558 thousand samples. They did not require much pre-processing on this data as it had no missing values and outliers. For feature selection, they used correlation between features to drop eliminate correlated features from the dataset using a threshold. Thus, using 166 features for training. They used a neural network architecture with Batch Normalization [19] and Dropout [20] layers in Keras. Multiple performance metrics were used for evaluation which includes Mean Absolute Error, Mean Squared Logarithmic error, Mean Squared Error, Root Mean Squared Error, Max Error and R2 as primary metric.

Random Search Cross-Validation was used to optimize the parameters of network. A higher R2 score would suggest better generalization. The tuned model achieved an R2 score of 0.8683 which was significantly higher than other methods tested by the authors. The other methods were Two-Stream Gated Recurrent Unit Network, Hybrid time series predictive neural network, LSTM, Multi Layer Perceptron (MLP), and Decision Trees.

Another approach towards this dataset was shared in the post titled "AutoEncoder and multitask MLP on new dataset(from Kaggle Jane Street)" by the user jrai [21]. The author suggests using de-noising auto-encoder and concatenating latent space to noisy inputs for training regression model.

2.3 Deep Learning related to the research

Zhang et al.[22] experimented with multiple tests to inspect the reasons for generalization in deep neural networks. Generalization gap refers to the difference between model's performance metrics on training and unforeseen testing data. It is argued that understanding the differentiating factor between the neural networks that generalize well and the ones that do not, would allow to build more explainable neural networks as well as introduce new systematic approach to robust model architectures.

Multiple randomization tests were performed to see if neural networks can fit random data. Varying levels of label corruption to images of ImageNet [12] and CIFAR10 [23] labels were applied where the model achieved 0 training error. Furthermore, they replaced images with completely random noise and the model was still able to overfit on the random images.

They also experimented with the effect of regularization on the neural network training. Most of the experiments were performed without explicit regularization. Here, explicit regularization refers to Data Augmentation, Weight Decay, and Dropout. They concluded that it may help but is not necessary for lower generalization error.

Zhang et al. [22] presented a framework to estimate the "effective capacity" of the model. They concluded that a model with more parameters than the sample size can be represented by a two-layer network.

Vaswani et al [7] introduced the scaled dot-product attention along with the Transformer architecture. TabNet [6] utilized Transformer architecture and attention mechanism. This approach allowed the sequential data to be processed parallelly unlike LSTM which processes data one step at a time. This architecture is the foundation on then state-of-the-art models like BERT [24], GPT [25]in natural language processing tasks. It is also being used in Computer Vision models with Vision Transformer [26]. The transformer is an encoder-decoder model where input is combined with positional encoding and masks. Both the encoder and decoder blocks contain multiple layers of Multi-Head Attention (MHA), a feed forward layer, normalization layers and skip connections. It uses key, query and value pairs to learn there to attend. Multi-Head

Attention has multiple heads and linear mappings to learn where to attend to the input.

Deep Generative models like Variational Auto-encoders (VAE) [27], Generative Adversarial Networks (GAN) [28] have shown impressive performance in producing realistic data especially in Computer Vision. The synthetic data can be used to overcome the shortage of training data. While the VAE uses real data to train, the GAN uses an adversarial set-up of a Generator and a Discriminator. Thus, the generator does not need to look at real data. There have been many variations of GAN framework like W-GAN [29], Conditional GAN [30], CTGAN [31]. CTGAN is a conditional GAN used to generate synthetic tabular data. Xu et al. [31] claimed that the CTGAN approach performed better than other deep learning approaches.

They mentioned multiple challenges in generating synthetic tabular data like different data types, non-Gaussian distributions, multi-modal data, and sparse and imbalanced data. Mode-specific normalization was introduced, which uses Gaussian Mixture Models (GMM) to estimate the number of modes and then normalize based on that. Unlike traditional GAN, where the generator does not account for imbalanced data, Conditional GAN uses conditions on the generator to generate data close to the distribution of real data. Combined with this, they introduced training-by-sampling to account for imbalance in the categories of training data.

Python library Synthetic Data Vault [32] has implemented this model, along with other statistical methods like Gaussian Copula.

2.4 Latent Representation

Auto Encoder consists of an encoder and a decoder where the encoder tries to compress the input to a much lower dimensional intermediate representation and the decoder learn to reconstruct the input from this latent vector representation. This unsupervised pre-training is often used to improve the training process during task specific fine-tuning. It can be beneficial to generative models as well.

Vincent et al. [33] proposed an approach to improve the robustness of these latent

representations. Before passing the input to the autoencoder, the input is corrupted, and the decoder is trained to reconstruct the original uncorrupted input in an end-to-end model. Vincent et al [33] argued that these denoising autoencoders can then be used for initializing other deep models.

MNIST [34] and different variations of it were used to test the effectiveness of this approach. Some transformations applied on the MNIST image classification dataset were, but not limited to, rotation, randomizing background, and combinations of these transformations. The trained stacked denoising autoencoder with 3 hidden encoder layers was used as initialization for the classifier and compared against SVM with Gaussian and polynomial kernels, Deep Belief Networks. A simple auto-encoder without input corruption was also trained, and it was found that the stacked denoising autoencoder outperformed other approaches by a significant margin.

We discussed auto-encoders and their effective use as feature extractors. The latent representations learned using unsupervised pre-training can provide good initialization for the fine-tuning task. Introduced by Chen et al., SimCLR [35] uses contrastive learning to improve the latent representations. It augments the images using multiple transformations and tries to generate embeddings using a non-linear projection layer. The representations from the augmented images using ResNet-50 are optimized to be similar; On the other hand, representations from different image pairs are optimized to be different. Cosine similarity is used as a similarity measure for latent representations. It uses NT-Xent loss to train on the image pairs. On ImageNet, it performed better than earlier self-supervised techniques.

To summarize, We found that tree-based models perform satisfactorily on most tabular data. Some studies claim that deep learning is catching up with tree-based models in this aspect. However, the transformers have presented a way to process the time series data in a parallel manner, unlike RNN, which does that sequentially. Deep learning provides the flexibility of a custom loss function and techniques like auto-encoders to reduce the data to a lower dimension. De-noising auto-encoders can help in improving latent representations of noisy data.

Ensemble of prediction is another approach that can combine predictions from

multiple predictors. A smart ensemble of multiple predictions often outperforms underlying models on multiple metrics.

CHAPTER 3

The Data

The dataset used for this thesis is from the Numerai weekly data science competition. The competition went through multiple changes in the last few years. There is a new round opening each week where new live data is released. The latest, as of November 5, 2022, is round 347.

The data is encrypted historical data of the stock market with anonymous feature names. Since the predictive structure is not lost in the encryption process, it is possible to find patterns and learn from the data. It is a time-series data with samples grouped into "eras". Eras suggest relative timing of samples. Meaning era 1 is older than era 2 and so on. There have been multiple versions of the data starting from 21 features and 5 targets to 1191 features and 20 targets in the latest version. This chapter discusses the structure of this time-series data and the progression from the second version to the latest version to get a clear understanding of the data and the competition.

The only unique values in all the versions of this dataset are from the set {0.0, 0.25, 0.5, 0.75, 1.0} for all features and targets except some targets in version 4; however, their distribution changes. For ease of processing, each feature name starts with the prefix "feature_" and targets with the prefix "target_". Apart from the features and the target columns, there are three more columns: id, era, and data_type. The described statistics of the datasets are calculated till round 347's data when we have a total of 1051 eras in total.

Three primary splits in the data are training, validation and live. Training and

validation sets are used for training the models. Each week, a new round opens, and a new live set, without targets, is made publicly available. The participants make predictions on the live set and submit it back to Numerai. Over the course of next four weeks, these predictions are evaluated in the global stock market. Based on the performance of those predictions, the participants get scored. The performance metrics are discussed in the next chapter.

The primary scoring function is Spearman's ranked correlation between predictions and the live market situation. If the predictions are positively correlated, the participants get positive score and negative if otherwise. In this thesis, we used the latest version of the data which is also the largest one.

3.1 Second iteration of the dataset

The second iteration of the dataset, V2, was released in late 2019 and is called the "Legacy Dataset". It had 310 features and one target. Here, all the unique values are from the set {0.0, 0.25, 0.5, 0.75, 1.0}. Some of the anonymized feature names were as follows: {'feature_constitution40', 'feature_wisdom38', 'feature_constitution111', 'feature_wisdom15', 'feature_constitution7', 'feature_constitution101', 'feature_charisma41', 'feature_constitution45', 'feature_strength24', 'feature_strength8',... }.

Training Data: 310 columns (features)

500K Rows

id	era	data_type	feature_..1	feature_..2	feature_..45	...	feature_..46	target_kazutsugi
n1..c9	era1	train	0	0	0.75	...	0.25	0.25
n2..08	era10	train	0.75	0.5	1	...	1	0.5
n5..69	era100	train	0.75	0.5	0.75	...	0.5	0.5
n8..5b	era101	train	0.25	0.75	0	...	0.25	0.25
n6..c2	era102	train	0.5	0.75	1	...	1	0.5
n8..a4	era103	train	0.25	0.5	0	...	1	0.25
n4..14	era104	train	0.75	0.75	0	...	0.25	0
n1..2e	era105	train	0.5	0	0.75	...	0.75	0
nc..f5	era106	train	0.75	1	1	...	0.25	1
n3..b7	era107	train	0.75	0.25	1	...	0.25	0.25

(a) Training

Tournament Data: 310 columns (features)

100K Rows

1.5M Rows

5K Rows

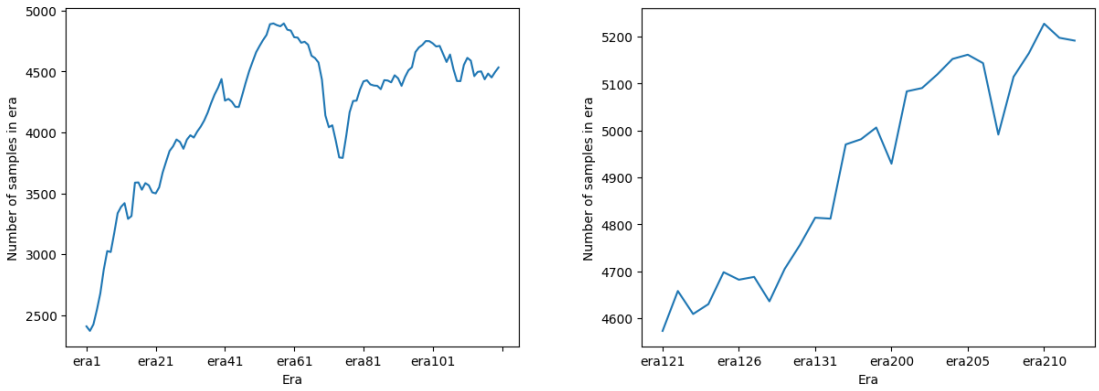
id	era	data_type	feature_..1	feature_..2	feature_..45	...	feature_..46	target_kazutsugi
n0..c2	era121	validation	0.25	0.75	0	...	0	0
n0..3f	era121	validation	0.75	0.5	0.5	...	0.5	0.25
nf..bb	era206	validation	0.25	0.5	0.75	...	0.75	0.75
nf..2e	era206	validation	0.25	0.5	0.5	...	0.5	1
n0..43	era575	test	0.25	0.5	0	...	0	nan
n0..ac	era575	test	0.5	0.5	0.5	...	0.25	nan
nf..4a	era920	test	0.75	0.75	0	...	0.25	nan
nf..3e	era920	test	0.5	0.5	1	...	1	nan
n0..8f	eraX	live	1	1	0	...	0.25	nan
n0..67	eraX	live	1	0.5	0.25	...	0	nan

(b) Validation

Fig. 3.1.1: Sample tables from V2

This data is split into four types, namely training, validation, test, and live. Only the train and validation split have targets. The test data is used for the purpose of back-testing after submitting predictions to the tournament, and live is the latest era for which we need to make predictions. The training and validation splits are static, while live data is appended to test data after the new round. There are 501808 samples in the training data with 314 columns, of which 310 are features, 1 target and the other three are id, era, and data_type, respectively. It is divided into 120 eras. Era 55 has 4893 samples which is the highest sample per era in training data. The lowest sample size of any era is in era 2. The average samples per era are 4181.73. The validation split has 28 eras and 137779 samples. In the Validation split, the maximum number of samples of any era is 5227 at era 210, while the minimum is on era 121

with 4573 samples. The test split has 2130083 samples with 310 features without associated targets, so it cannot use that split for the purpose of training. Fig. 3.1.2 shows the number of samples per era for training and validation data, respectively.



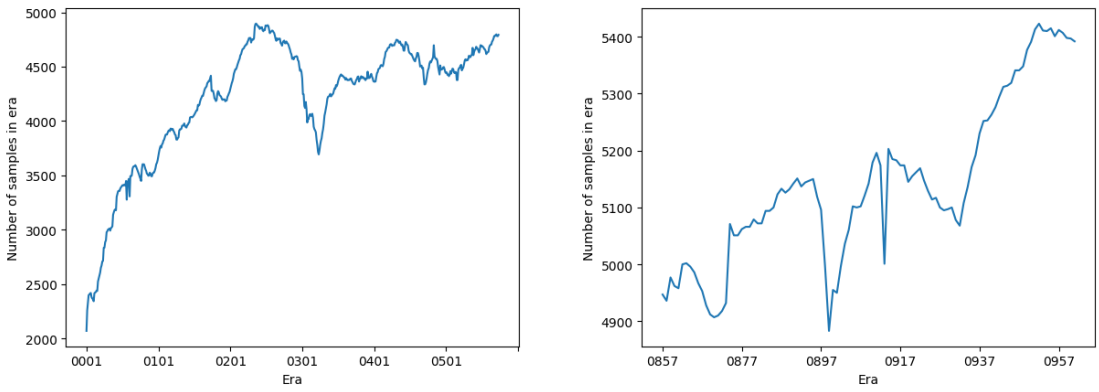
(a) Training

(b) Validation

Fig. 3.1.2: Samples per era in training and validation splits in version 2 of the dataset

3.2 Third iteration of the dataset

The third iteration of the dataset, V3, is called the "Super Massive Dataset" and was released in September 2021. It introduced major changes in the dataset format with more features, targets, and samples within more eras. This supermassive dataset is split into three parts: training, validation and live. Only the training data is static. After each round, live data is appended to the validation data once the live targets are calculated. This means the validation data keep expanding over the course of time.



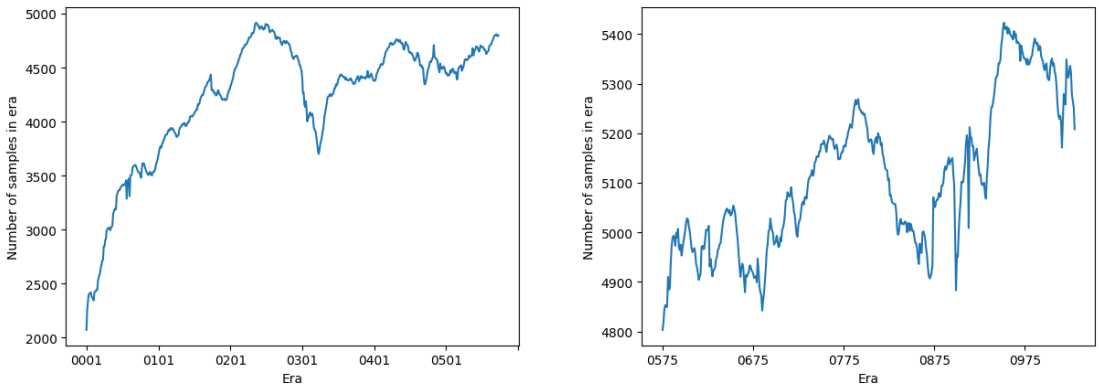
(a) Training (b) Validation

Fig. 3.2.1: Samples per era V3

It has 1050 features, 20 targets, and 2412105 samples grouped into 574 eras in training data. This is a significant increment over the previous version regarding the information available. The maximum samples in any training era are 237 with 4896, and the least in era 1 with 2070. The average samples per era increased by 21 samples. The validation split has 105 eras.

3.3 Fourth iteration of the dataset

The latest version, V4, is the largest one, with 574 training eras and more than 4 times the increment in the validation eras. It was released in April 2022. It has 1191 features. Target columns are the same in both V3 and V4. As the validation data keeps expanding, we decided to use validation data until era 1030.



(a) Training (b) Validation

Fig. 3.3.1: Samples per era in V4

era	id	dat...type	fea...mite	fea...inze	fea...eman	fea...hala	...	tar...4_20	tar...4_60	tar...4_20	tar...4_60	tar...4_20	tar...4_60
0001	n003bba8a98662e4	train	1.00	0.50	1.00	1.00	...	0.25	0.25	0.25	0.25	0.50	0.25
0001	n003bee128c2fcfc	train	0.50	1.00	0.25	0.75	...	1.00	1.00	0.50	0.50	0.75	1.00
0002	n0009c7a6acb5c1d	train	0.25	0.50	0.50	0.50	...	0.50	0.75	0.50	0.75	0.75	0.50
0002	n004caed2f26ae3c	train	1.00	1.00	0.00	1.00	...	0.50	0.50	0.50	0.50	0.50	0.75
0003	n00019f22179a91d	train	0.00	0.25	0.75	0.50	...	0.50	0.75	0.50	0.50	0.50	0.50
...	...	...	...	...	...	...	...	...	...	...	...	...	...
0572	n000d2593be5cb9e	train	0.00	0.00	0.00	0.00	...	0.75	0.75	0.75	0.75	0.75	0.75
0573	n00000fb07d026be	train	0.50	1.00	0.00	1.00	...	0.50	0.75	0.50	0.75	0.50	0.75
0573	n00067fafc63e4f3	train	0.50	0.50	0.00	0.75	...	0.75	0.50	0.75	0.75	0.75	0.50
0574	n00025c2020feed2	train	0.25	0.50	0.00	0.75	...	0.50	0.75	0.75	0.75	0.50	1.00
0574	n000366ab45f8145	train	0.50	0.75	0.50	0.75	...	0.25	0.25	0.25	0.25	0.25	0.25

1148 rows × 1206 columns

(a) Training

era	id	dat...type	fea...mite	fea...inze	fea...eman	fea...hala	...	tar...4_20	tar...4_60	tar...4_20	tar...4_60	tar...4_20	tar...4_60
0575	n000101811a8a843	validation	0.50	0.00	1.00	0.00	...	0.50	0.50	0.50	0.50	0.50	0.50
0575	n001e1318d5072ac	validation	0.25	1.00	0.50	0.50	...	0.00	0.25	0.25	0.25	0.00	0.25
0576	n0004e3727abc707	validation	1.00	0.75	0.00	1.00	...	0.25	0.25	0.50	0.25	0.25	0.25
0576	n0008ecf930b3229	validation	0.50	0.25	0.00	1.00	...	0.50	0.50	0.50	0.50	0.50	0.50
0577	n000be59a0bee991	validation	0.75	0.00	1.00	0.00	...	0.75	0.25	0.75	0.50	0.50	0.50
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1028	n000220cf5b0dd00	validation	1.00	1.00	1.00	1.00	...	0.50	NaN	0.50	NaN	0.50	NaN
1029	n0000d29e1b9a388	validation	0.25	0.00	0.50	0.25	...	0.50	NaN	0.75	NaN	0.50	NaN
1029	n00049a73abf71c3	validation	0.75	0.75	1.00	0.25	...	0.50	NaN	0.50	NaN	0.50	NaN
1030	n000752d4f3b391b	validation	0.25	1.00	0.25	0.75	...	1.00	NaN	0.75	NaN	1.00	NaN
1030	n0007eb8e8b92503	validation	0.50	0.75	0.50	0.75	...	0.50	NaN	0.50	NaN	0.50	NaN

912 rows × 1206 columns

(b) Latest Validation

Fig. 3.3.2: Sample tables from V4

Fig. 3.3.3 compares the number of eras in each split between different versions.

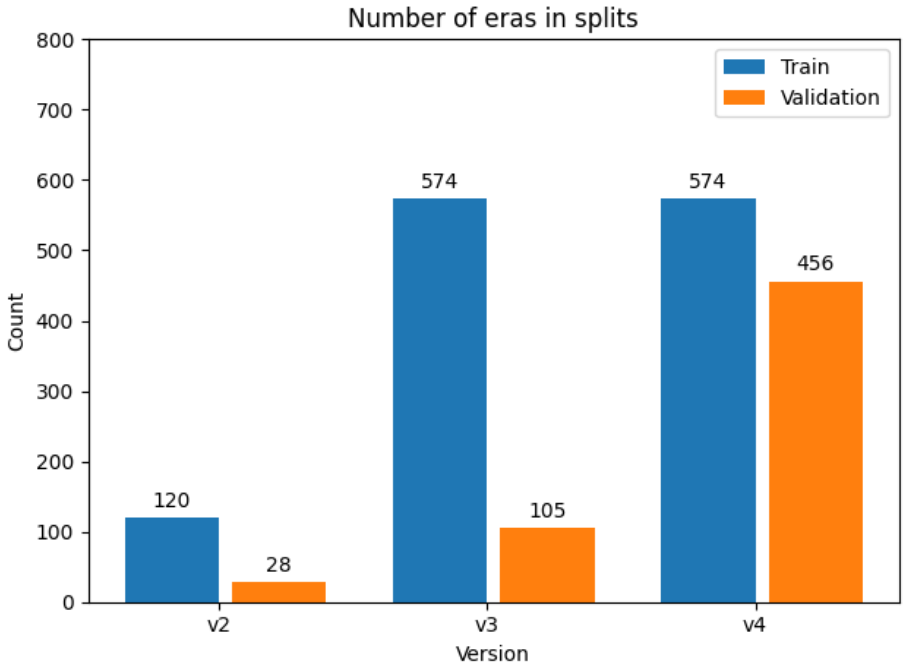


Fig. 3.3.3: Comparison of number of training and validation eras in three versions.

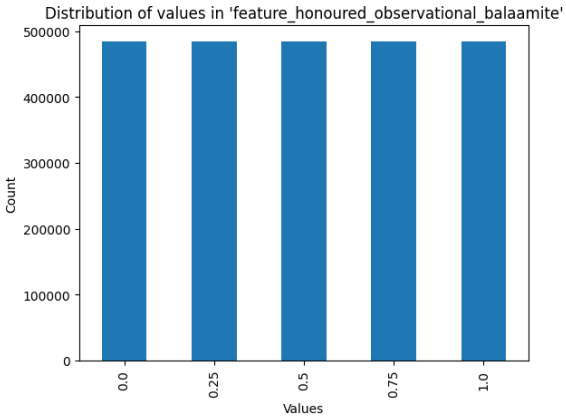
Version	Train and validation eras	Release	Features	Targets
V2	148	Late 2019	310	1
V3	679	September 2021	1051	20
V4	1030	April 2022	1191	20

Table 3.3.1: Comparison of data versions.

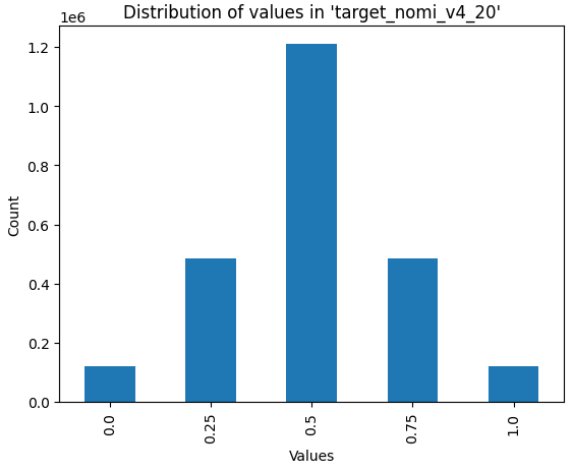
Fig. 3.3.4 shows the distribution of values in the dataset. Unlike most machine learning problems where data needs heavy cleaning before it's usable, this dataset is very clean. The values are already scaled without any null values. The features are distributed uniformly across those discrete values. The primary scoring target, 'target_nomi_v4_20' is very similar to a gaussian distribution. Another target variable, 'target_paul_v4_20' has seven unique values in similar distribution. Due to the huge size of rows and columns, the dataset comes with lists of two feature sets namely: 'small' and 'medium' to train on machines with relatively lower compute capabilities. The small and medium sets contain feature subsets with 38 and 472 features

respectively.

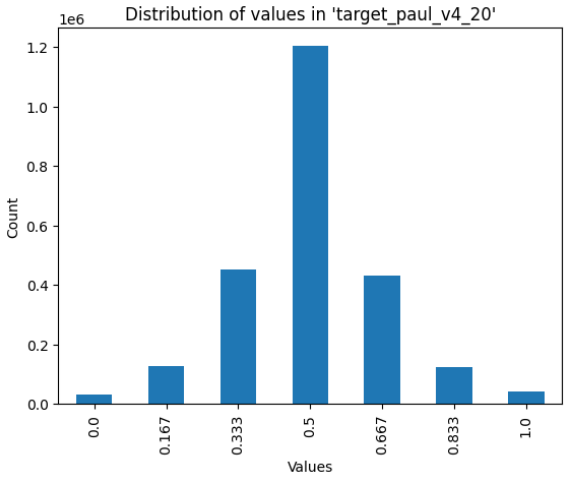
Due to the huge size of this dataset, another files are also provided with integer format to save memory. This integer format converts float values {0.0, 0.25, 0.5, 0.75, 1.0} to integers {0, 1, 2, 3, 4}.



(a) Features



(b) target_nomi_v4_20

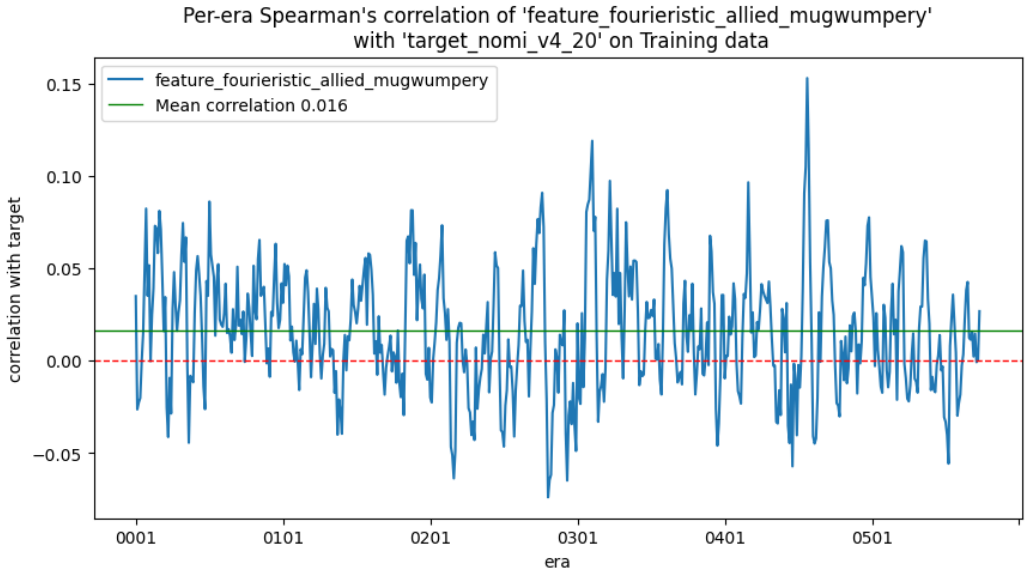


(c) target_paul_v4_20

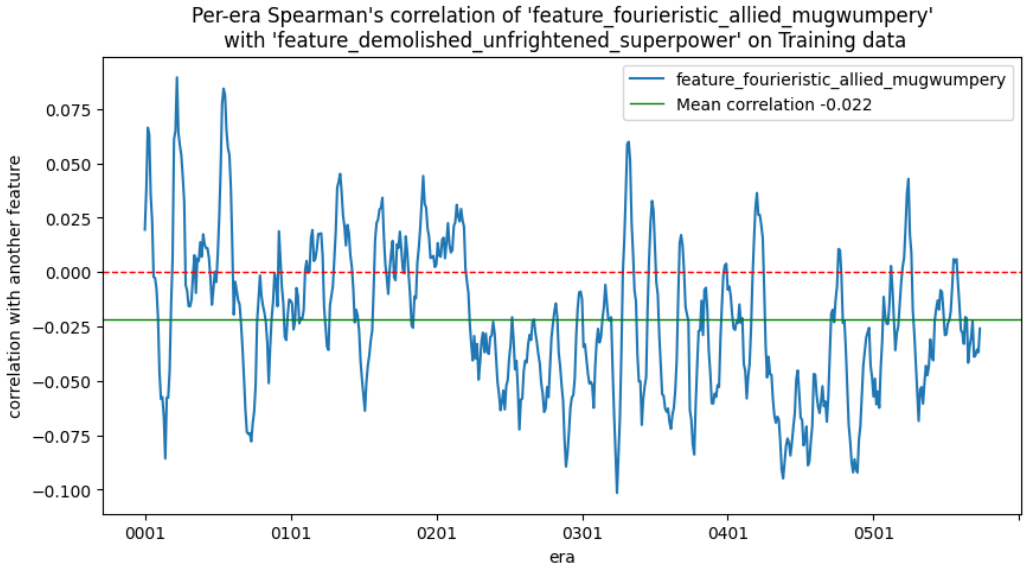
Fig. 3.3.4: Distribution of features and targets in data V4

Although the dataset is cleaned for modelling, what makes it complex is the nature of the features and size of the data. The features do not follow the same importance over time. Fig. 3.3.5 (a) shows the Spearman's correlation between a feature 'feature_fourieristic_allied_mugwumpery' and the target column. It should be

noticed that this feature does not follow the exact correlation to target across time. So, a simple Linear regression may not be able to capture this complexity of features. Not only features to target correlation, the correlation between features also changes over time as shown in Fig. 3.3.5 (b).



(a) Feature to target correlation



(b) Feature to feature correlation

Fig. 3.3.5: Spearman's correlation calculated amongst features and targets

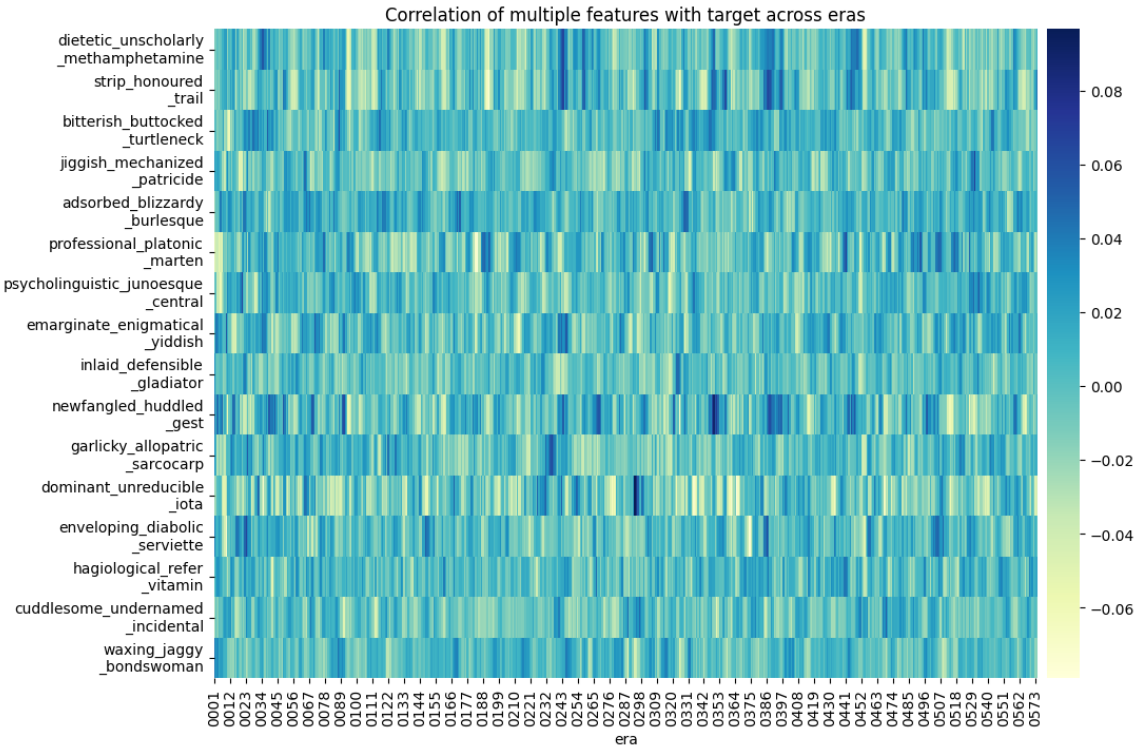


Fig. 3.3.6: Per-era correlation of some features to target

Some features may have similar correlation with targets in some eras and have no similarity in other eras. Fig. 3.3.6 shows the correlation between some features and target across the training eras. Thus, a naive model relying on some features can result into poor generalization on out-of-sample data leading to poor performance. In this competition, a model that is robust and generalizes well on out-of-sample data is desirable due to this dynamic nature of features.

CHAPTER 4

Machine Learning methods utilized

This chapter gives an overview about machine learning and the approaches that we used in the thesis. The simplest way to imagine machine learning is, an unknown function that maps two data points. Here, the task is to approximate that function using data. However, the key is to let the algorithm learn from the data rather than a human being manually fixing it. Over the course of iterations, the algorithm tunes itself to reduce the overall error and eventually converges to the point of estimation. Thus, it is possible to imagine it as an iterative process to reduce the error.

Machine Learning is a subset of a large research area called Artificial Intelligence (AI). It is broadly divided into three primary types based on the way an algorithm learns: Supervised learning, Unsupervised learning, and Reinforcement Learning.

4.1 Overview of Machine Learning

4.1.1 Supervised Learning

As the name suggests, Supervised learning is where we have a target (label) for every data point (sample) which we want to predict using a model. The learning process happens by first making a prediction and calculating how far the prediction is from the true target using a loss function. Then the model tries to reduce this error value

based on its learning method. The dataset we have is best suited for a supervised learning method as we have features and associated targets. Linear regression is one of the simplest supervised learning algorithms.

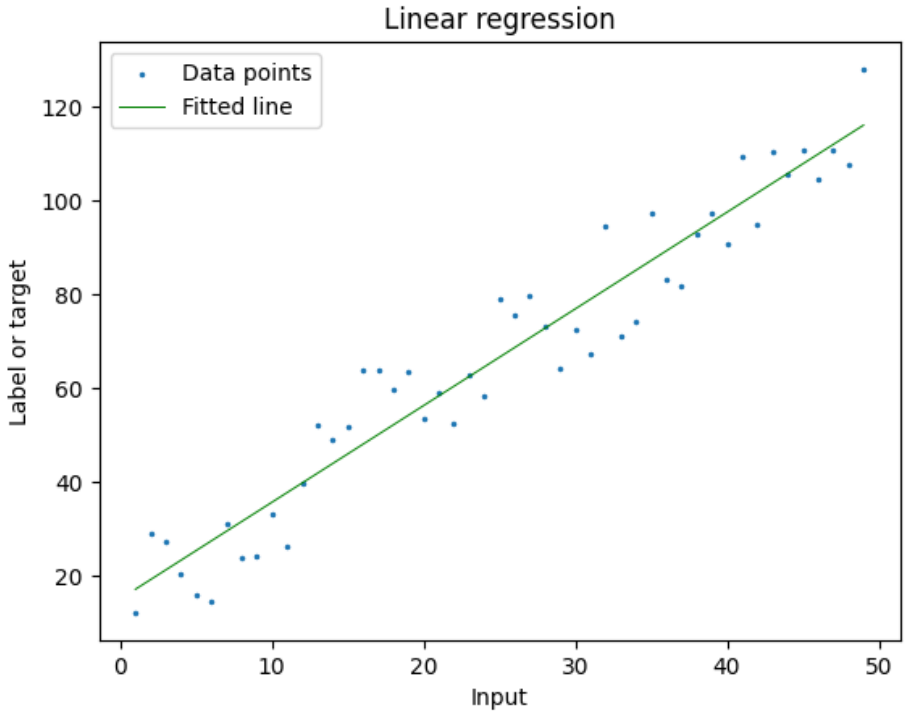


Fig. 4.1.1: Illustration of a Simple Linear regression

We start with a random parameter of a line and calculate the distance between prediction and target values. Ideally, we would like to reach to the state where error is zero, but it may not be possible in this case with linear regression as shown in Fig. 4.1.1. So, our goal is to reach to global minima of the loss value. Based on the targets available, it is possible to divide the supervised learning further into Regression and Classification. In the case of linear regression, we want to output a continuous value as prediction. So it falls under regression. Classification is where the prediction is expected to be one of many classes. The targets are often discrete such as in {0, 0.1, 0.2, 0.5}, {“car”, “truck”, “bike”}, {0, 1}.

4.1.2 Unsupervised Learning

While supervised learning algorithms learns by approximating the relationship between inputs and targets, unsupervised learning methods do not explicitly need labels to learn the patterns in the data. Clustering is an intuitive example of this. We let the model group the data into clusters where similar samples are assigned coordinates closer to each other. It is also useful in dimensionality reduction where the model tries to extract maximum information available in original input and compresses into a relatively smaller vector representation.

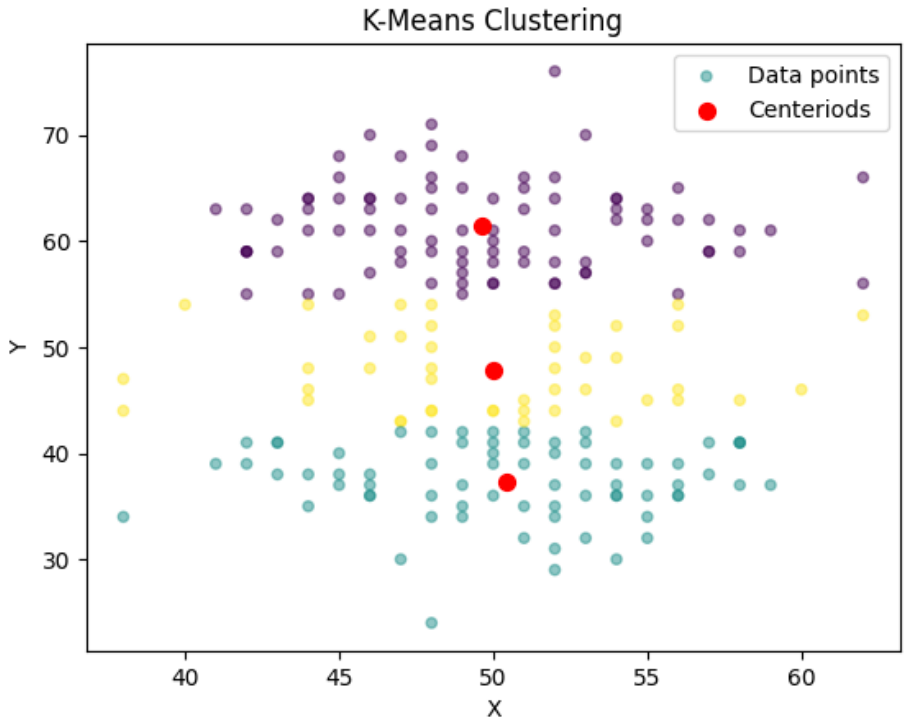


Fig. 4.1.2: Illustration of clustering on random data points

4.1.3 Reinforcement Learning

Reinforcement learning consists of a dynamic agent and environment setup where the agent interacts with the environment and receives feedback. The agent has an incentive in the form of a reward. If the agent performs as expected, it gets positive feedback in the form of a reward; this framework allows for training models with great degrees of freedom, such as robotic arms, genetic algorithms, simulations, etc.

4.2 Tree-based learning

Trees are an important data structure. They are hierarchical and have nodes and edges connecting the nodes. The very first node is called the "root" node. If two nodes are connected via an edge in two adjacent levels, the node at a higher level is called the "parent," and the lower-level node is called the "child" node. Except for the root node, all other nodes must have a parent node. The end nodes without child nodes are called "leaf" nodes. This is where that branch of the tree ends.

Decision trees [36], as the name suggests utilizes tree structure to learn decision making by asking question to the input on each node. The hierarchical nature allows the tree to add multiple queries and generate a decision on the leaf node in the end. Fig. 4.2.1 illustrates a simple Decision tree.

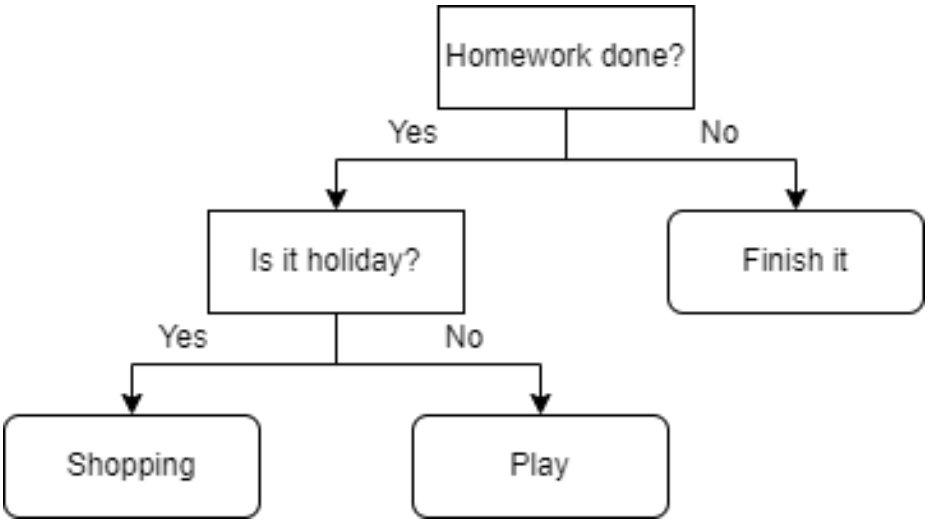
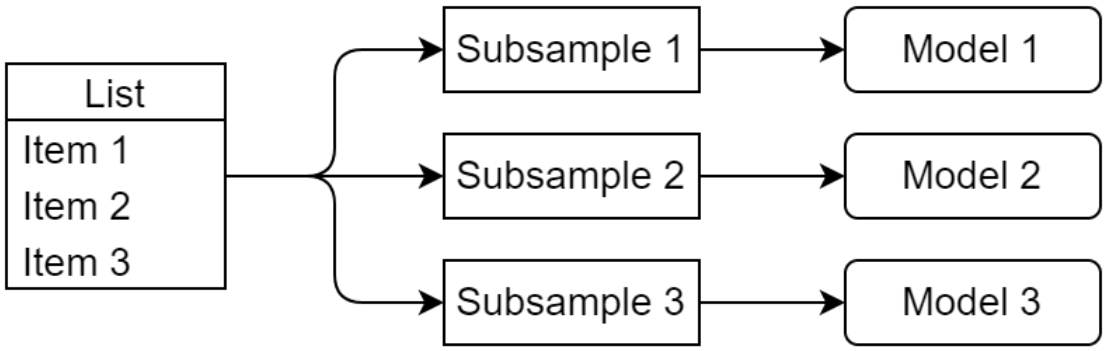


Fig. 4.2.1: Illustration of a Simple Linear regression

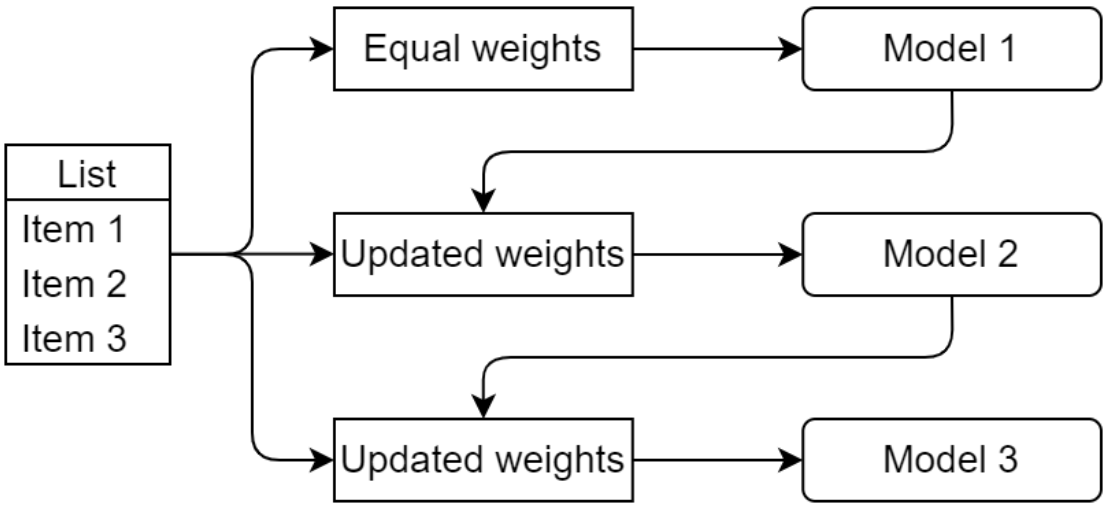
Tree models have become popular amongst ML practitioners. Their hierarchical decision-making nature makes them intuitive and easily interpretable. It is possible to inspect each node to understand which factors led to a specific prediction.

Ensemble learning is a way to combine multiple predictors to make the final prediction more accurate and robust. This also helps in generalized predictions. Two primary methods to build an ensemble of trees are Bagging and Boosting. Bagging is where each tree is built parallelly with subsample of data with replacement. On the

other hand, Boosting builds the ensemble sequentially. Each new tree is built upon to improve the predictions till last tree. Bagging aims to reduce overfitting while Boosting tries to reduce underfitting [37].



(a) Bagging



(b) Boosting

Fig. 4.2.2: Illustration of Bagging and Boosting

Random forest [38] is an ensemble method that uses multiple decision trees to build the final predictor. It uses random sampling of data while training. Some popular methods utilizing boosting are XGBoost [14], CatBoost [13], LightGBM [39], etc. They are highly optimized for big data handling and have huge community support.

4.2.1 XGBoost

XGBoost [14] is an open-source tree-based algorithm that was created in 2016 at the University of Washington as part of a research effort. Extreme Gradient Boosting is the methodology behind it, which is reflected in its name. The areas of regression, classification, ranking and custom prediction challenges are just some of the places where XGBoost may be used. Since that time, it has gained widespread adoption in the ML community. It has been a contributing factor in the victories of a significant number of Kaggle competitions throughout the years. It is compatible with various programming languages, including Python and Java, amongst others.

In addition, it supports a range of parameters like L1, and L2 regularization, tree depth, variations of boosting, and many more. It can utilize multiple cores on the CPU and GPU, thus training faster. Support for null values (NaN) values is also out-of-the-box.

4.2.2 CatBoost

Developed in 2017, CatBoost [13] is another boosting algorithm that is open-source and has wide community support. CatBoost claims to be very good at handling categorical values, thus, it is named Categorical Boosting. Unlike other tree-based ML libraries, CatBoost builds symmetric trees as it has some efficiency benefits. It uses ordered boosting, with weak learners trained on a random subset of data, to increase randomization and reduce overfitting. It is one of the first boosting libraries to have built-in support for categorical columns meaning there is no need to explicitly convert categorical columns to numbers before feeding them to the model. An impressive feature of CatBoost is the efficient utilization of GPU. This makes the training significantly faster. It has support for multiple types of tabular problems like Regression, Classification, and Ranking. It now also supports text data, so it is possible to use text as input to the model. In our experiments, CatBoost performed really well on GPU with maximum utilization, and more often than not, the default parameters worked really well. It has excellent support for visualizing the model as

well, which makes it very interpretable.

4.2.3 LightGBM

Like XGBoost and CatBoost, LightGBM [39] is also an open-source tree-based modelling tool that was developed by Microsoft. Although it supports GPU for training, its primary feature is the surprising training efficiency on multi-core CPU. Like CatBoost, we can feed categorical data to LightGBM simply by specifying the column index or casting it to the "Category" type. LightGBM uses Histogram based splitting and grows trees level-wise, unlike XGBoost, which grows trees leaf-wise. The efficiency on a multi-core CPU makes it an ideal candidate for training and predicting on any platform with or without GPU support.

4.3 Deep Learning

Deep Learning attempts to imitate the biological neural networks in the brain. It has multiple layers of interconnected neurons where the information flows from the input neuron to final neuron through the weighted connections. Perceptron [40] is a simple neural network. It uses back-propagation to gradually update the connections between the neurons in such a way that the final output is closest to the given target in a supervised training set-up. This is done by first passing the input through model and generating a prediction. An error value between the prediction and the actual label is then calculated along with the gradients of weights with respect to the error value. The weights are then updated in the direction of lower error value. An iteration of this process over the entire set is often called an "epoch". This requires all the operations to be differentiable. It is called Deep Learning because of the number of stacked layers in the model architecture, the more the layers, deeper the network. Neural networks need the input in a numerical form. Thus, it adds an extra step of manual numerification of input during pre-processing. Means the inputs like text, images, must be converted to a numerical representation before feeding them to a neural network.

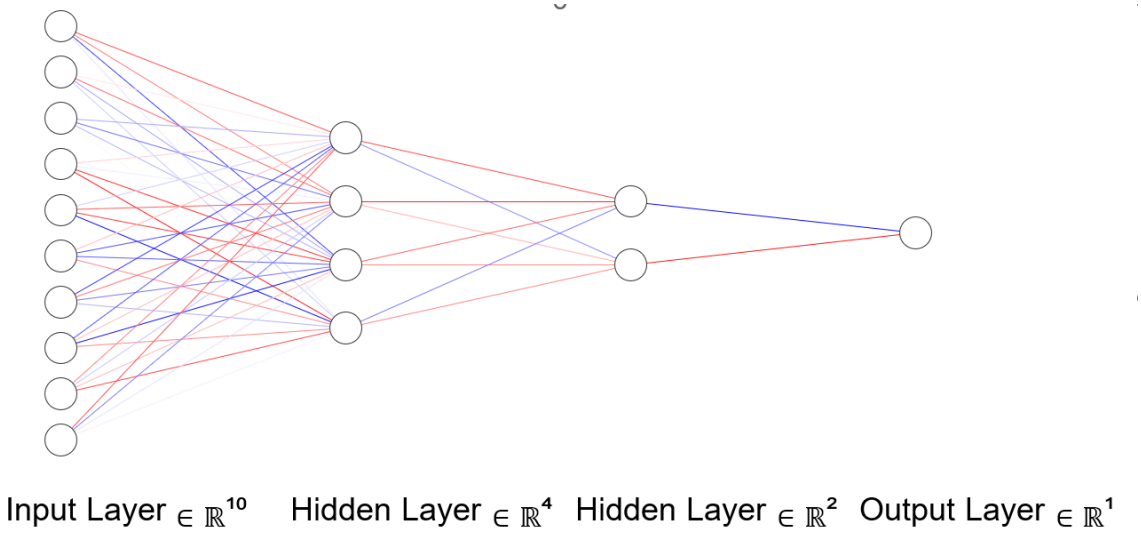


Fig. 4.3.1: Illustration of a simple Multi-layer Perceptron with two hidden layers

An impressive feature of neural networks is that they learn the function using the weights and non-linearity and can find some very complex patterns in the data which would otherwise not be possible with linear models. There are numerous studies suggesting that neural network can approximate any function given enough capacity as discussed in previous chapter [22]. However, due to this complex nature of the learning process and representing information internally, neural networks are often hard to interpret. With the availability of massive parallel computing, deep learning has shown applications in variety of fields from Computer Vision, Natural Language Processing, to even generating synthetic images, audio, and texts. In supervised pre-training, a multi-layer neural network model M is trained on a task A. We can use the trained model M's initial few layers as an initialization for task B. This is called *transfer learning*.

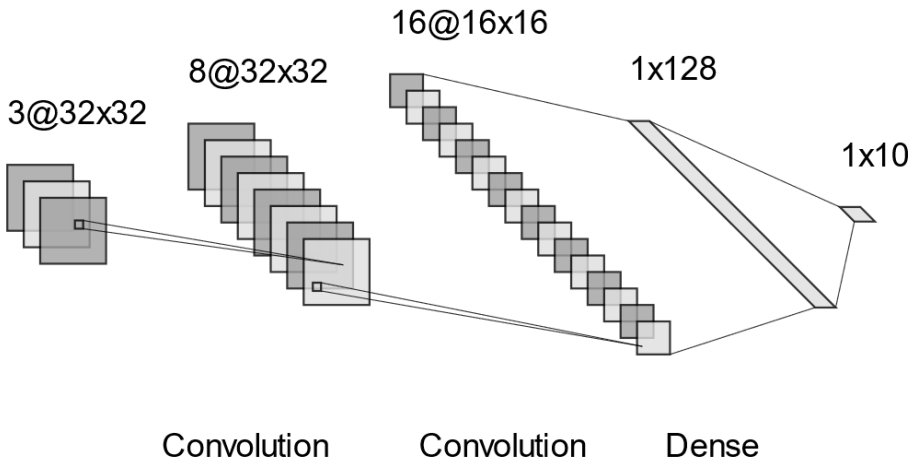


Fig. 4.3.2: Illustration of a Convolutional neural network with linear layers at end

In unsupervised pre-training, usually, an encoder-decoder model is trained. The encoder has multiple layers connected, which reduces the latent dimension size and doubles as input to the decoder, as shown in Fig. 4.3.3. The decoder then tries to reconstruct the input. Thus, the model is constrained to extract as much information as it can into the smaller latent vector. Once trained, the encoder is often used as an initializer to other models, or it can work as a dimensionality reduction step.

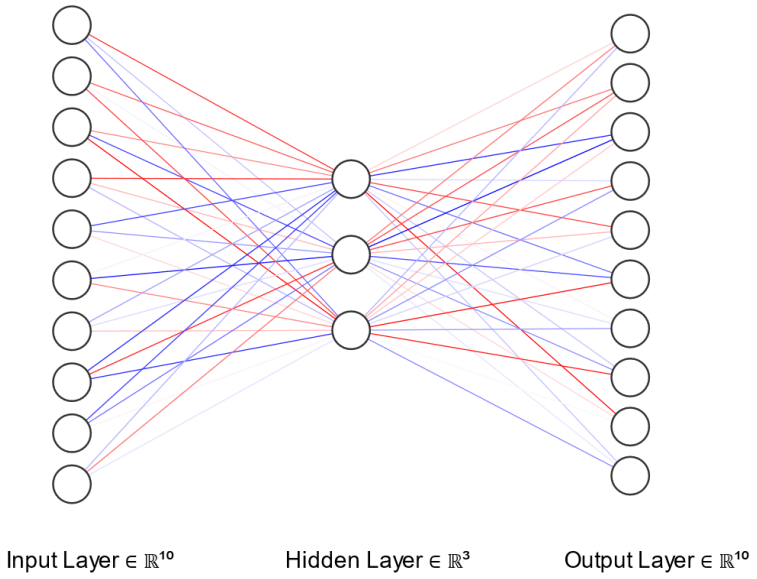


Fig. 4.3.3: Denoising Auto-Encoder input and channels, with reconstructed output.

Another interesting direction of research is on latent space. Enhancing the latent representations in neural networks is an ongoing active research topic. SimCLR [35], Supervised Contrastive Learning [41] and similar techniques attempt to make latent representations different for different classes and similar for similar classes. For example, two images of similar-looking cats should have similar latent representations and should not be similar to representations of dogs. This idea allows the model to learn diverse patterns from the data. We built upon this idea of less similar representations and de-noising auto-encoder.

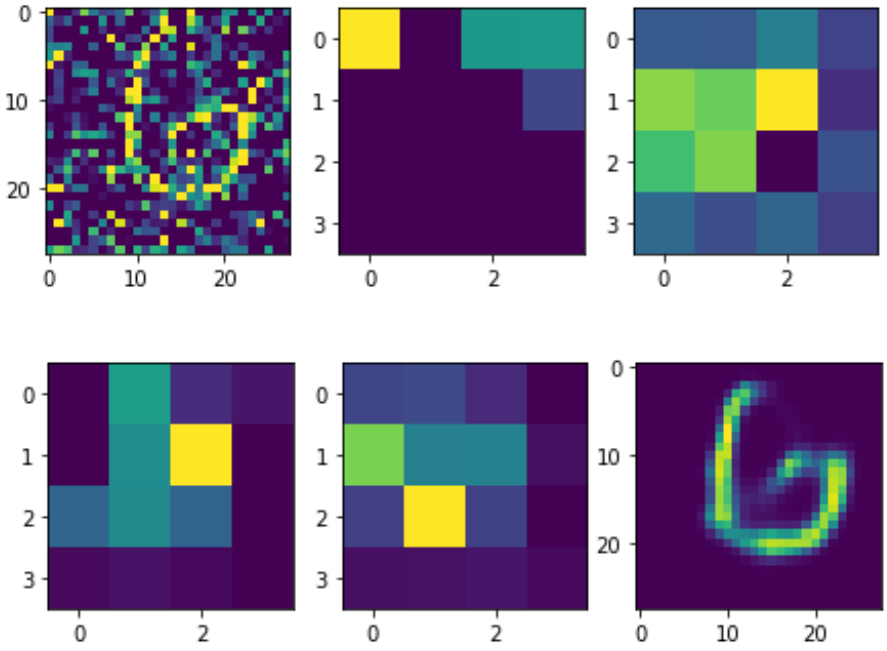


Fig. 4.3.4: Illustration of a de-noising auto-encoder

4.4 Performance Measures

In the iterative process of Machine Learning, we need a way to measure the progress and quality of learning. ML algorithms have an objective function that is used to optimize the model's parameters and is often referred to as the *Loss* function. However, the loss function has a different purpose than performance metrics. While loss functions are usually differentiable in nature, the metrics may not be. Different metrics are used based on the nature of the problem. Some are suited better for classification, and some for regression. Here, we discuss some metrics we used to measure and compare models' performance for regression.

4.4.1 Mean Squared Error

Mean Squared Error (MSE) is a popular objective function and metric used in optimizing machine learning models. It is defined as the mean of squared error between target and predicted values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{1}$$

where:

N = Numer of samples

$\hat{x}_i$ = Predicted value for sample on index i

x_i = True value for sample on index i

where N is number of samples, $\hat{x}$ is predicted value and x is true value.

It penalizes the predictions that are far away from the ground truth. The sign is removed by taking the square of the difference. Thus, it results in a positive value. Since it is differentiable, it can be used in optimizing neural networks.

4.4.2 Root Mean Squared Error

Root Mean Squared Error (RMSE) is simply the square root of MSE. While MSE squares the error to penalize large differences and turn the values to positive, RMSE takes the square toot of MSE to scale it down to original units from the squared values. The ideal RMSE score is 0, which is unlikely unless the problem is simple. It is often used in tree models as the default objective function.

4.4.3 Pearson's correlation

Pearson's correlation coefficient, also referred to as "the correlation coefficient," is a measure of linear correlation between two variables. It ranges between [-1, 1]. A correlation coefficient of 1 suggests two perfectly positively linearly correlated variables, while a correlation coefficient of -1 suggests negatively correlated variables. It is formulated as the ratio of covariance to the product of standard deviations of two variables.

$$\rho = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} \tag{2}$$

where:

cov = Covariance

X, Y = variable

Another formulation is as below.

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{3}$$

where:

n = Sample size

x_i, y_i = Samples at index i

$\overline{x}, \overline{y}$ = Mean of variables

Since this is differentiable, it is possible to use this function as an objective or loss function to optimize a deep learning model.

4.4.4 Spearman's correlation

While Pearson's correlation measures the linear correlation between two variables, the Spearman's correlation coefficient measures the rank correlation between two variables. It is calculated like Pearson's correlation except that the variables are ranked before calculating covariance and standard deviation. This operation of ranking the variables is not differentiable which makes it not suitable for a loss function in a neural network training.

$$r_s = \rho_{R(X),R(Y)} = \frac{\text{cov}(R(X), R(Y))}{\sigma_{R(x)} \sigma_{R(y)}} \tag{4}$$

where:

$\text{cov}(R(X), R(Y))$ = covariance of ranked variables

$\sigma_{R(x)}, \sigma_{R(y)}$ = standard deviation of ranked variables

Another popular formula can be applied if all n are unique integers.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{5}$$

where:

$d_i = R(X_i) - R(Y_i)$ = covariance of ranked variables

n = number of samples

This is the primary metric in the dataset. The data is grouped into eras and the Spearman's correlation between target and ranked prediction is calculated. The payouts in each round are based on this score.

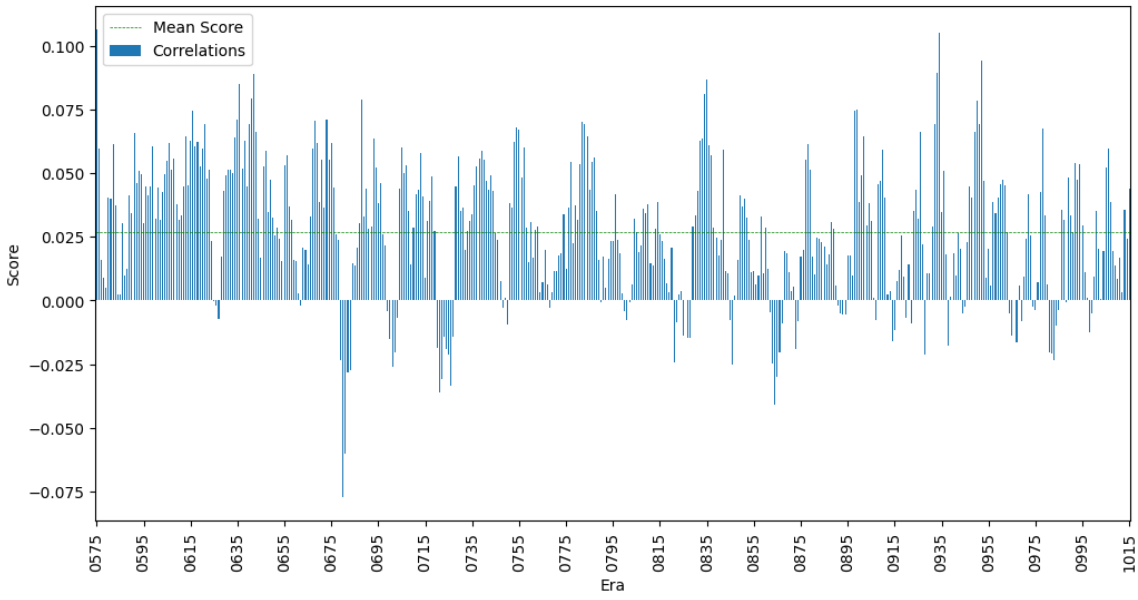


Fig. 4.4.1: Illustration of per era correlation

4.4.5 Maximum Drawdown

Maximum Drawdown indicates how much our portfolio can lose from the last peak value [42]. Since, our returns are based on Spearman's correlation. Maximum drawdown is calculated as,

$$NumeraiMaxDD = \max(\frac{era\_corr - last\_peak\_corr}{last\_peak\_corr}) \tag{6}$$

4.4.6 Numerai Sharpe

Sharpe ratio measures the risk-free returns of the portfolio. Since our predictions are used to build a portfolio by ranking the rows in an era, we can calculate how good our model is based on the Sharpe ratio. The higher the ratio, higher the returns per risk. This suggests the returns per risk. Sharpe ratio in Numerai's context is defined as:

$$NumeraiSharpe = \frac{mean(per\_era\_correlations)}{std\_dev(per\_era\_correlations)} \tag{7}$$

where:

$$per\_era\_correlations = \text{Spearman's rank correlation between predictions and targets in each era}$$

The primary metric here is Spearman's correlation, which is used for scoring in the competition. It is also used to calculate the Numerai Sharpe ratio, which suggests returns per risk, suggesting long-term performance. Thus, we aim to improve performance on Numerai Sharpe while keeping a higher mean correlation across eras with lower maximum drawdowns on out-of-sample eras.

CHAPTER 5

Methodology

This chapter describes the architecture of the proposed framework for extracting useful features from the input and using them on extended estimators to enhance the provided baseline model. The training happens in two parts, first, is a neural network de-noising auto-encoder trained end-to-end with different latent and regression heads. Then the latent representation is concatenated to train a tree-based model. Tree-based models have shown excellent performance on tabular data. The dataset used for this thesis has a very high noise-to-signal ratio. Instead of comparing deep learning approaches to tree-based models, we propose an architecture or a framework that combines the best of both worlds to achieve diverse and good predictions.

5.1 Cross-Validation

Although we have millions of rows in the dataset, it is explicitly split into training and validation data. For evaluating multiple models using cross-validation, only the training split with 574 eras was used. The given validation split is used as a hold-out set, often referred to as 'test_set' to measure out-of-sample performance. As suggested by Lopez De Prado[5], the K-Fold cross-validation fails in finance mainly for the reason of leaking of future data. So, we used an expanding window era splitting approach as illustrated in Fig. 5.1.1 for generating training and evaluation folds for cross-validation. Although we do not have a timestamp for each sample and era, the

eras represent relative timestamps meaning era 1 is older than era 2 and so on.

Expanding window approach with dropping 12 eras between the train and evaluation set was used. The training eras were split into 3 folds as illustrated in Fig. 5.1.1. The reasoning behind 3 fold and an embargo of 12 eras was to have a higher number of eras, and dropping 12 eras between splits suggests an approximate interval of 12 weeks, thus, resulting in balanced splits of the dataset.

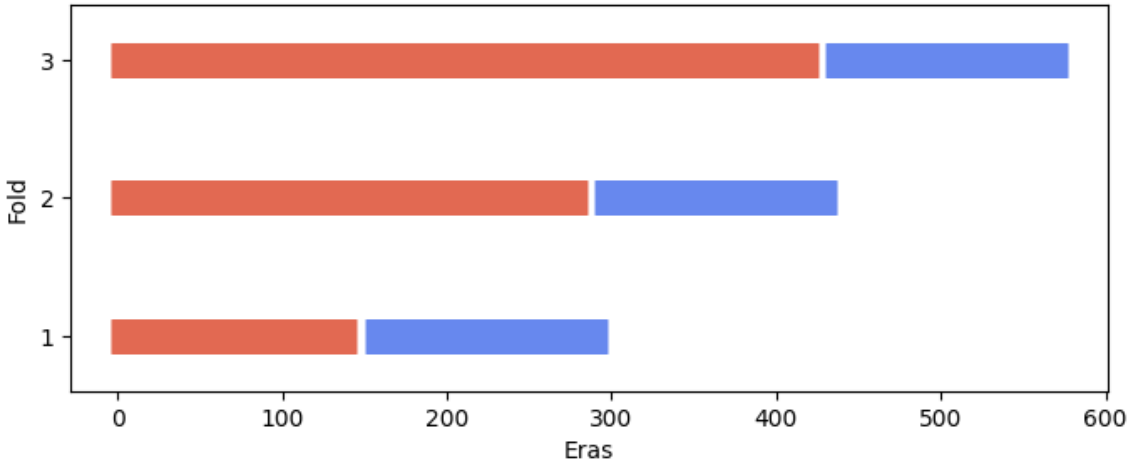


Fig. 5.1.1: Illustration of cross-validation splits

5.2 Architecture

The architecture of the proposed framework is illustrated in Fig. 5.2.1. It consists of a de-noising Auto-Encoder model with multiple regression heads and a custom loss function.

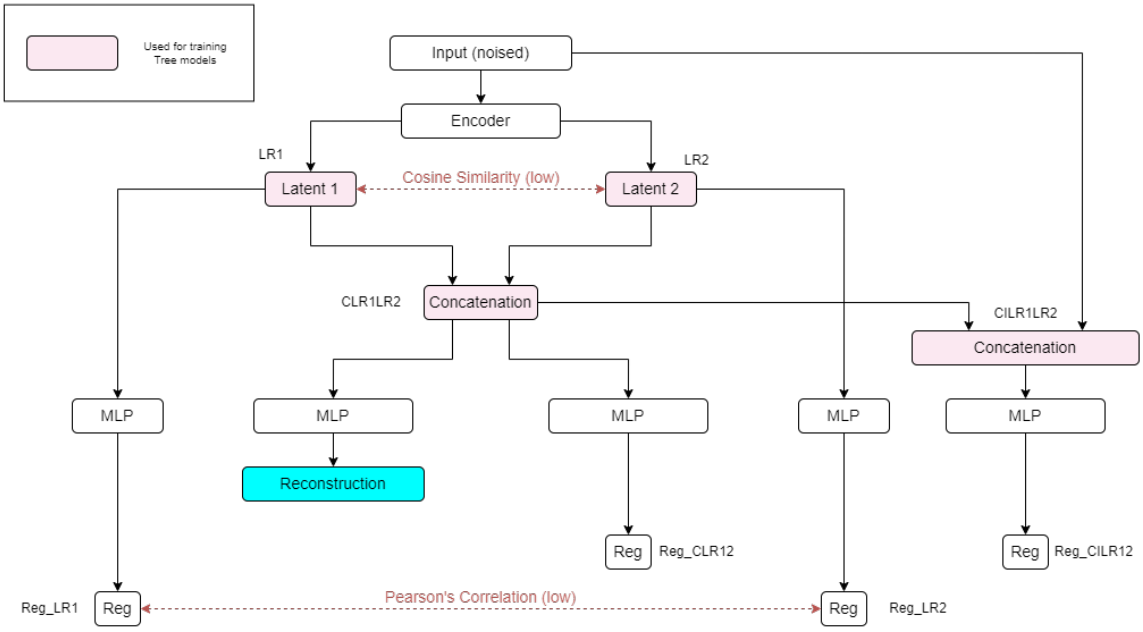


Fig. 5.2.1: Proposed model architecture

5.2.1 Input corruption

Instead of setting a fixed batch size for training, a dynamic batch size was chosen. Each batch is based on era, thus, each training and validation era is used as a batch for training and evaluation.

The input to the de-noising auto-encoder model is corrupted. For targets at the decoder, input without corruption is used. For the corrupted input, randomly selected features are inverted after each epoch. This helps in making the encoder-decoder more robust. For example, a feature (column) with values [1, 0.25, 0.5, 1, 0] will become [0, 0.75, 0.5, 0, 1]; thus, its correlation is inverted. This randomness in data augmentation should suggest to the model not to over-fit on certain features. This way, the model is expected to reconstruct the original input from the corrupted one on the decoder. Similar to data augmentation in Computer Vision, this corruption of input acts as regularization for the model.

5.2.2 De-noising auto-encoder

The auto-encoder is a multi-layer fully-connected deep neural network. The encoder has a decreasing number of neurons in each layer, while the decoder has an increasing number of neurons as we go deeper into the network. The Encoder splits into two separate layers LR1 and LR2. These layers' representations are constrained to be orthogonal to each other. The absolute value of Cosine similarity between them is added to the final loss function of the model. Thus, we penalize the model if these two representations are similar to each other for a given sample. The idea behind this is that we create two ways for the data to flow in the deep neural network, but they should be orthogonal, leading to unique latent representations from the same encoder. This helps in generating diverse encoding. While being orthogonal to each other, LR1 and LR2 are both constrained to learn orthogonal vectors individually as well. The loss function between LR1 and LR2 is defined as,

$$latent\_similarity(x, y) = \frac{1}{N}\sum_{i=1}^{N} |cos\_sim(x_i, y_i)| = \frac{1}{N}\sum_{i=1}^{N} |\frac{x_i \cdot y_i}{||x_i|| \cdot ||y_i||}| \quad (1)$$

where:

$cos\_sim(x_i, y_i)$ = Cosine similarity between $x_i and y_i$

x_i, y_i = Two latent representations for a sample at index i

These latent representations are then concatenated, generating a representation CLR1LR2 or concat(LR1, LR2). This is used as the input to the decoder to reconstruct original input. Mean Squared Error between original input and reconstructed input is used as loss function for optimization. Along with decoder, the concat(LR1, LR2) is fed to another MLP head as well. The model uses Batch normalization [19] and Dropout [20] layers. The concatenated LR1, LR2 are concatenated with the noisy input to generate CILR1LR2 or concat(corrupted_input, LR1, LR2).

The loss function for the reconstruction head is the Mean Squared Error be-

tween the original un-corrupted feature values and the reconstructed values at reconstruction head. The reason for selecting MSE is to get the reconstructed values as close to original values as possible.

5.2.3 Regression heads

The four latent representations: LR1, LR2, CLR1LR2, and CILR1LR2 are used to train regression heads on a given target. These neural network-based regression heads are trained using a combination of MSE and Pearson's correlation as a loss function. The MSE between predictions and targets should be low, while Pearson's correlation should be high. The similarity between regression heads on top of LR1 and LR2 are penalized using Pearson's correlation if it exceeds a threshold. This should help in generating diverse predictions as well as latent representations.

The same four latent representations are then used to train tree-based models. Thus, a total of 8 predictions are generated with varied levels of diversity. These predictions are then used to build an ensemble using a Random Forest model.

5.2.4 Loss function

With deep learning, it is possible to use a custom loss function depending on the problem's requirement. As long as the function is differentiable and it is possible to calculate the gradients of weights with respect to the loss value, we can use any arbitrary loss function for optimization.

The final loss function is a combination of multiple loss functions utilized to train the neural network in an end-to-end manner. Since Pearson's correlation is differentiable, It is used in combination with MSE to train regression heads. In addition, the regression heads $Reg_{LR1}, Reg_{LR2}, Reg_{CILR12}$ are trained using a combination of MSE and correlation defined in the equation 2. This forces the model to reduce MSE between predictions and targets while increasing correlation between them.

$$regression\_heads\_loss = MSE(y\_true, y\_pred) - Corr_P(y\_true, y\_pred) \quad (2)$$

where:

$Corr_P$ = Pearson's correlation between two variables

Thus, the final loss for the neural network becomes,

$$\begin{aligned} Loss = (&latent\_similarity(LR1, LR2)+ \\ &Corr\_abs_P(Reg_{LR1}, Reg_{LR2}, Threshold)+ \\ &reconstruction\_loss+ \\ &regression\_heads\_loss) \end{aligned} \quad (3)$$

where:

$Corr\_abs_P$ = Absolute value of Pearson's correlation between two variables exceeding a threshold value

$reconstruction\_loss$ = MSE between un-corrupted input and predicted reconstruction at reconstruction head

$regression\_heads\_loss$ = regression loss at heads $Reg_{LR1}, Reg_{LR2}, Reg_{CILR12}$

5.2.5 Ensemble predictions

The proposed system has 8 regression heads, out of which 4 are as shown in Fig. 5.2.1 while other 4 are tree-based regressors trained on latent representations. Another meta-predictor is trained using these 8 regressor predictions. Thus, we end up with 10 predictions. However, the ones with most information are tree models trained on concatenated latent representations of concat(LR1, LR2) and concat(corrupted_input, LR1, LR2) along with meta-predictor.

CHAPTER 6

Experiments and Results

This chapter discusses the baseline model and experiments using the proposed methodology and briefly touches on the experiments with generative models. The proposed methodology was evaluated using 3-fold validation using the splitting method explained in section 5.1. Between the training and evaluation split, a gap of 12 eras was kept. The same models were then trained on the entire 'training' split and evaluated on the 'validation' split on the 'medium' feature set to get a fair comparison with the baseline model.

6.1 Experimental Setup

The results for all the experiments on the baseline model and proposed framework were collected on the same hardware configuration. We've used a gaming laptop equipped with a 2.30 GHz Intel Core i7-11800H octa-core processor, 64 GB system RAM, a 4 GB NVIDIA GeForce RTX 3050 Laptop GPU, and Windows 10 operating system.

The programming language Python with Anaconda distribution (3.8.13) is used in Jupyter notebook format using Visual Studio Code IDE. Python library Pandas (1.5.1) is extensively used for the purpose of reading the data and saving intermediate Python objects using Pickle format. Numerapi is used to download different versions of the dataset and other helping files. Numpy and Joblib are used to perform

vectorized operations and parallelization, respectively.

Python implementation of XGBoost (1.6.2), LightGBM (3.3.3) and CatBoost (1.1) are used to build tree-based models. TensorFlow's (2.9.0) Keras API is used to build deep neural network models. For the experiments with generative models, Google Colab was used since it provides better GPU resources.

6.2 Baseline Model

Along with the data, a baseline model script is also provided within the same folder. The purpose of the baseline model is to allow users without much data science experience to train a model quickly. This script uses LightGBM to train a model on the provided 'medium' feature set with 472 features. It trains on the entire training data. The model with the same parameters is used as the baseline model in the experiments.

6.3 Using Medium Feature set

For comparison with the baseline model, we used the provided 'medium' feature set to train and evaluate out methodology along with other tree-based models like Histogram based Gradient Boosting Decision Trees, XGBoost, CatBoost, with similar parameters to baseline and default parameters. We wanted to include all the features in training and let the auto-encoder learn to compress the feature space. Our methodology produces 8 predictions in total; 4 from regression heads and 4 from tree models trained on latent representations. However, not all those predictions are used here. 'latent_cated' is a tree model trained on concatenated representations of LR1 and LR2. 'latent_full' is a tree model trained on concatenated representation of the input, LR1 and LR2. 'latent_meta' is an ensemble of all the regression heads of the proposed model. At the same time, 'latent_mean' is a simple ranked average of predictions.

For experiments, we performed 1) cross-validation on provided 'training' data of

574 eras, 2) Trained on provided 'training' data of 574 eras, evaluated on provided 'validation' data which gives a performance on hold-out set, and 3) cross-validation on entire data of all eras.

The metrics used to compare the performance are mean correlation across eras, standard deviation of scores across eras, Numerai Sharpe ratio and maximum drawdown. Maximum drawdown here represents how much lower our scores drops from the last peak. Thus, a value of -0.15 means 15% drop in scores from last peak.

6.3.1 Cross-Validation on training split

In this particular experiment, we used the provided training split to perform cross-validation with 3 folds. The data was divided using eras. The training eras were divided into 3 folds with a gap of 12 eras between training, and evaluation splits as illustrated in the Fig. 5.1.1. The Tables 6.3.1 and 6.3.2 show the performance of tree-based models and our method on cross-validation in-sample and out-of-sample, respectively.

Model	Mean corr	Std dev	Sharpe	Max DD
HistGBDT	0.189418	0.025533	7.585323	0
XGBoost	0.349072	0.024178	14.337692	0
CatBoost	0.359840	0.026125	13.753306	0
LightGBM (baseline)	0.233890	0.027003	8.717857	0
latent_cated	0.153000	0.027090	5.662020	0
latent_full	0.186672	0.027526	6.768404	0
latent_meta	0.250943	0.026702	9.407828	0
latent_mean	0.173153	0.028670	6.058977	0

Table 6.3.1: Cross-validation in-sample results on 'medium' set.

Model	Mean corr	Std dev	Sharpe	Max DD
HistGBDT	0.046778	0.028989	1.631147	-0.098909
XGBoost	0.035191	0.022792	1.570647	-0.073783
CatBoost	0.044947	0.028983	1.596611	-0.120406
LightGBM (baseline)	0.052501	0.031142	1.726779	-0.108057
latent_cated	0.040065	0.022423	1.824583	-0.026383
latent_full	0.043760	0.023761	1.887324	-0.045384
latent_meta	0.051127	0.030027	1.749567	-0.105935
latent_mean	0.044029	0.024543	1.819415	-0.036795

Table 6.3.2: Cross-validation out-of-sample results on 'medium' set.

Amongst the tree-based models in cross-validation, it can be seen that tree models have way higher in-sample performance compared to out-of-sample predictions. This suggests significant over-fitting. XGBoost and CatBoost showed higher in-sample performance. Proposed methodology achieves similar results with better generalization.

6.3.2 Provided training and validation sets

To further evaluate our method on hold-out data, we trained all the models on provided training data and evaluated on provided validation set. This is similar to a typical machine learning evaluation strategy where data is split into training, evaluation and testing. Here, validation split acts as testing split to measure the performance on completely unforeseen data.

Model	Mean corr	Std dev	Sharpe	Max DD
HistGBDT	0.124669	0.028011	4.450723	0
CatBoost	0.249026	0.025345	9.825329	0
XGBoost	0.234253	0.022418	10.449129	0
LightGBM (baseline)	0.150411	0.028289	5.316921	0
latent_cated	0.116180	0.025239	4.603227	0
latent_full	0.135838	0.025492	5.328647	0
latent_meta	0.164361	0.027402	5.998104	0
latent_mean	0.127138	0.027036	4.702624	0

Table 6.3.3: Scores on provided training set

Model	Mean corr	Std dev	Sharpe	Max DD
HistGBDT	0.024348	0.029351	0.829548	-0.236339
CatBoost	0.027780	0.027273	1.018587	-0.178210
XGBoost	0.022055	0.025950	0.849884	-0.180163
LightGBM (baseline)	0.027577	0.030875	0.893178	-0.217328
latent_cated	0.026456	0.027107	0.975971	-0.204608
latent_full	0.027079	0.027147	0.997512	-0.199861
latent_meta	0.027267	0.029717	0.917552	-0.202845
latent_mean	0.027606	0.028596	0.965363	-0.214069

Table 6.3.4: Scores on provided validation set

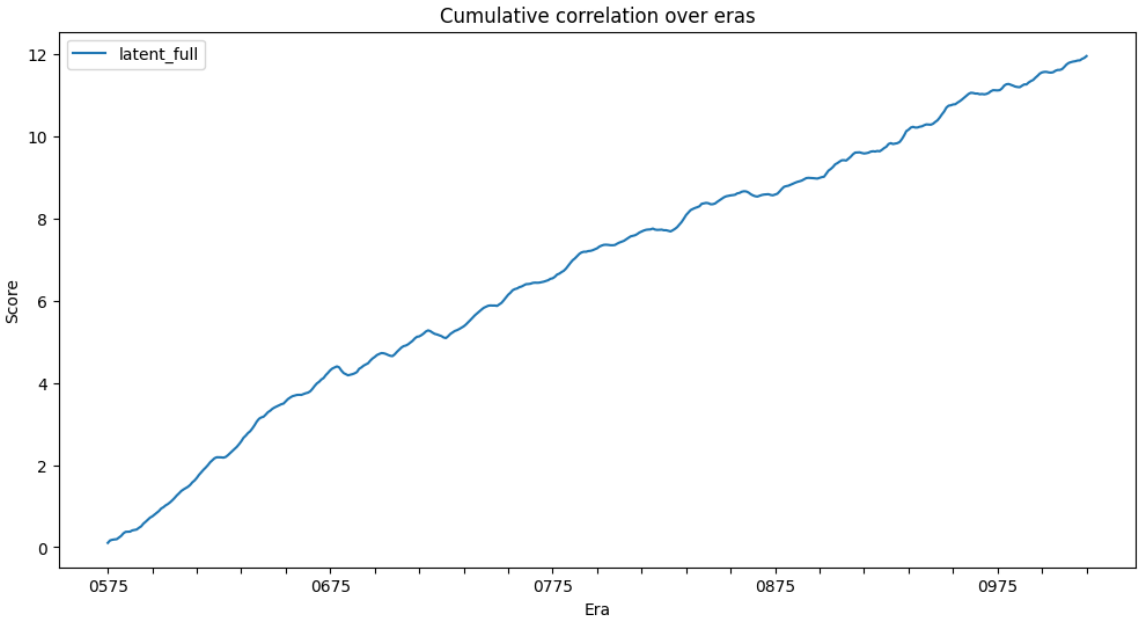


Fig. 6.3.1: Cumulative Correlation scores on validation data

6.3.3 Cross-validation on full set

Extending our cross-validation experiments, we concatenated both training and validation sets to create a full set for cross-validation. The baseline model and predictions from proposed methods were compared. Similar results were noticed where proposed method with 'latent_full' outperformed baseline model on the basis of Sharpe ratio in out-of-sample performance with slight drop in mean correlation across eras and lower max drawdown. This suggests a robust, relatively better generalized model over baseline.

Model	Mean corr	Std dev	Sharpe	Max DD
LightGBM (baseline)	0.173102	0.027760	6.352010	0
latent_cated	0.123497	0.026481	4.658308	0
latent_full	0.148048	0.026704	5.528855	0
latent_meta	0.173695	0.027697	6.383770	0
latent_mean	0.135429	0.027505	4.933936	0

Table 6.3.5: Cross-validation on full data: In-sample

Model	Mean corr	Std dev	Sharpe	Max DD
LightGBM (baseline)	0.038567	0.030702	1.246133	-0.213263
latent_cated	0.032886	0.026182	1.269613	-0.164400
latent_full	0.035233	0.026838	1.315669	-0.165907
latent_meta	0.038500	0.030691	1.244406	-0.211854
latent_mean	0.034459	0.027260	1.281429	-0.165843

Table 6.3.6: Cross-validation on full data: Out-of-sample

6.4 Synthetic data generation

Along with latent feature concatenation, we briefly experimented with synthentic data generation. To generate synthetic eras, CTGAN [31] was used. Due to the large number of feature columns, the provided ‘small‘ feature set was used which has 38 features. To further reduce compute overhead, every fifth eras was sampled from training data. After training CTGAN model for 500 eras, It did not show any significant improvement in out-of-sample performance. ‘preds_extended_full‘ is trained on synthetic data along with provided training data while ‘preds_baseline‘ is trained only on provided training data. It is noticeable that the synthetically generated when concatenated to the original data did not help in improving the

performance.

Model	Mean corr	Std dev	Sharpe	Max DD
preds_extended_full	0.220256	0.027911	7.891242	0.0
preds_baseline	0.283448	0.025786	10.992122	0.0

Table 6.4.1: Results on training set for synthetic data

Model	Mean corr	Std dev	Sharpe	Max DD
preds_extended_full	0.014463	0.024344	0.594137	-0.292441
preds_baseline	0.017773	0.023060	0.770695	-0.217692

Table 6.4.2: Results on validation set for synthetic data

CHAPTER 7

Conclusion and Future Work

Our results indicate that unsupervised learning used to extract and compress information from complex tabular data can be used to enhance the performance of a tree-based baseline model. With a slight reduction in mean correlation, a noticeable improvement in the Sharpe ratio suggests higher returns per given risk with a reduction in the drawdown as well. We have demonstrated that the proposed framework surpasses the baseline on both in-sample and out-of-sample performance validated through cross-validation. To extend this framework, the learned feature vector can be used to train a multitude of models for the ensemble as well. For example, there can be multiple Latent heads instead of two trained on a custom loss function. An extension of this system can be multiple regression heads constrained to reduce the correlation between each other to generate a diverse base for a smart ensemble.

One drawback of our approach is the computation, as we need to train a neural network in an end-to-end manner with a complex loss function. However, as the training data is static, we need to train the system only once.

Our experiments to generate synthetic eras using a smaller feature set using CTGAN did not show any significant improvement. For future work, better generative models like Diffusion models can be tested for the purpose of data generation using more computing.

Bibliography

[1] C. Banton, *Alpha vs. Beta: What's the Difference?* [accessed 2022-11-25], 2022. [Online]. Available: `https://www.investopedia.com/ask/answers/102714/whats-difference-between-alpha-and-beta.asp`.

[2] *Investment process*, [accessed 2022-10-15]. [Online]. Available: `https://numer.ai/fund`.

[3] M. Nabipour, P. Nayyeri, H. Jabani, A. Mosavi, and E. Salwana, "Deep learning for stock market prediction," *Entropy*, vol. 22, no. 8, p. 840, 2020.

[4] S. Basak, S. Kar, S. Saha, L. Khaidem, and S. R. Dey, "Predicting the direction of stock market prices using tree-based classifiers," *The North American Journal of Economics and Finance*, vol. 47, pp. 552–567, 2019.

[5] M. L. De Prado, "Advances in financial machine learning," in John Wiley & Sons, 2018.

[6] S. Ö. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 6679–6687.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022.

[9] S. Popov, S. Morozov, and A. Babenko, "Neural oblivious decision ensembles for deep learning on tabular data," *arXiv preprint arXiv:1909.06312*, 2019.

[10] L. Katzir, G. Elidan, and R. El-Yaniv, "Net-dnf: Effective deep modeling of tabular data," in *International Conference on Learning Representations*, 2020.

[11] b., *Github - baosenguo/Kaggle-MoA-2nd-Place-Solution*, [accessed 2022-11-17], 2021. [Online]. Available: `https://github.com/baosenguo/Kaggle-MoA-2nd-Place-Solution`.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[13] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: `https://proceedings.neurips.cc/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf`.

[14] 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*, 2016, pp. 785–794.

[15] S. Thakur, S. Sharma, *et al.*, "Forecasting stock price using conditional inference tree," in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, IEEE, 2018, pp. 591–595.

[16] S. Singh and S. Sharma, "Forecasting stock price using partial least squares regression," in *2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, IEEE, 2018, pp. 587–591.

[17] B Vasudevan *et al.*, "Effective implementation of neural network model with tune parameter for stock market predictions," in *2021 2nd International Confer-*

ence on Smart Electronics and Communication (ICOSEC), IEEE, 2021, pp. 1038–1042.

[18] K. Singh, R. Tiwari, P. Johri, and A. A. Elngar, "Feature selection and hyperparameter tuning technique using neural network for stock market prediction," *Journal of Information Technology Management*, vol. 12, no. Special Issue: The Importance of Human Computer Interaction: Challenges, Methods and Applications. Pp. 89–108, 2020.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] jrai, *Autoencoder and multitask MLP on new dataset (from Kaggle Jane Street)*, https://forum.numer.ai/t/autoencoder-and-multitask-mlp-on-new-dataset-from-kaggle-jane-street/4338, [Online; accessed 2022-11-17], 2021.

[22] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, *Understanding deep learning requires rethinking generalization*, 2016. DOI: 10.48550/ARXIV.1611.03530. [Online]. Available: https://arxiv.org/abs/1611.03530.

[23] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[25] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[27] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[29] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, 2017. DOI: 10.48550/ARXIV.1701.07875. [Online]. Available: https://arxiv.org/abs/1701.07875.

[30] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[31] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[32] N. Patki, R. Wedge, and K. Veeramachaneni, "The synthetic data vault," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2016, pp. 399–410.

[33] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[34] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[35] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, PMLR, 2020, pp. 1597–1607.

[36] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[37] soumya7, *Bagging vs Boosting in Machine Learning - GeeksforGeeks*, [accessed 2022-11-17], 2019. [Online]. Available: `https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning`.

[38] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[40] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[41] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 661–18 673, 2020.

[42] A. Hayes, *Maximum Drawdown (MDD) Defined, With Formula for Calculation*, [accessed 2022-11-22], 2022. [Online]. Available: `https://www.investopedia.com/terms/m/maximum-drawdown-mdd.asp`.

Vita Auctoris

NAME:	Surajsinh Prakashchandra Parmar
PLACE OF BIRTH:	Navsari, Gujarat, India
YEAR OF BIRTH:	1999
EDUCATION:	
	Gujarat Technological University, B.E. in Computer Engineering, Surat, India, 2020
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2022