Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis International Bachelor Econometrics and Operations Research

Volatility Forecasting for S&P500 Index Returns: A Dynamic Ensemble Approach using Neural Networks and Random Forests with Synthetic Data Integration

Daris Fadhilah (595255)

Supervisor:	dr. Hakan Akyuz
Second assessor:	dr. Xiaomeng Zhang
Date final version:	1st July 2024

Abstract

Accurate volatility forecasting is important for effective risk management and the creation of robust financial strategies. This study extends upon the work of Lee and Kang (2024), which proposed the approach of a dynamic ensemble of regression neural networks. This study particularly explores the effectiveness of this approach within the context of financial time-series forecasting. To combat the issues of data scarcity and noise in data-scarce nodes, we advance this approach by integrating synthetic data generated using a Gaussian copula model fitted to represent the same data distribution as the training set of each node. The proposed extension aims to increase data availability and provide denoised training data, thereby reducing the risk of overfitting and improving generalization. Our experiments focus on forecasting the volatility of S&P 500 index returns. The results show that the new approach performs well under stable market conditions with low volatility. However, it fails to outperform traditional baseline models such as GARCH(1,1) and RiskMetrics during periods of rapid volatility changes. We also adapted this approach to a random forest model, which showed beneficial results in ensembling the predictions. This shows that the proposed approach can be adapted to other model architectures to enhance predictive accuracy. Nevertheless, models that utilize synthetic data integration demonstrate promising improvements, although their statistical significance remains marginal. While the current findings do not show significantly better performance, future improvements to this methodology, aimed at creating more representative synthetic data, could prove useful not only within the domain of financial forecasting but also for other data-intensive fields such as healthcare and insurance.

Contents

1 Introduction

Accurate financial forecasting is important for portfolio managers and policymakers to optimize portfolio performance and create effective economic policies. Improved forecasting leads to better decision-making, which therefore would improve economic and strategic outcomes for financial and economic institutions. Therefore, advancing research on financial forecasting methodologies is essential for continual improvement in financial performance. Additionally, the benefits of advancements in ensemble prediction methodologies extend beyond finance. These techniques have widespread applications across numerous fields, as evidenced by the widespread adoption of ensemble methods, such as Random Forests, in various industries. This broad applicability underscores the importance of continued research and development in ensemble forecasting techniques.

Data security and privacy concerns often make transmitting information between different nodes unfeasible. This limitation necessitates novel techniques to combine predictions across models in various nodes without sharing data. This study builds on the idea proposed by Lee and Kang (2024) to create a privacy-aware dynamic ensemble within the context of regression neural networks. The practical applications of such an approach span various domains and industries such as finance, telecommunications, and insurance.

While the concept of ensembling is not new, it has not been widely explored within the context of regression neural networks (Lee & Kang, 2024). One aspect that previous research does not explore is the effectiveness of this approach on different data distributions between nodes. Additionally, existing studies do not examine regressions within the context of sequential time-series data, which is the focus of this study.

Time series data, particularly in the field of finance, present unique challenges that demand careful consideration to prevent overfitting. Financial data are characterized by a high noise-to-signal ratio, stemming from their inherent stochastic properties. This characteristic poses some challenges for predictive modeling, especially when using highly flexible model structures such as neural networks. The susceptibility of neural networks to overfitting in noisy environments necessitates an exploration of alternative modeling approaches. Random Forests, with their simpler structure and inherent robustness, present a potentially viable alternative. Their ensemble nature and decision tree base may offer advantages in handling the complex, noisy patterns inherent in financial time series. To further address the challenge of overfitting, particularly in scenarios with limited or noisy training data, we propose the integration of synthetic data into the training process. This approach aims to augment the training set, potentially improving model generalization and reducing overfitting tendencies. These considerations lead us to our primary research question:

Can the implementation of a dynamic ensemble approach and synthetic data integration effectively reduce model overfitting in the context of financial time-series forecasting?

In contrast, traditional financial modelling, which primarily relies on econometric techniques, has remained largely unchanged for decades. Although machine learning has been rapidly adopted in other industries, its integration into finance has been slow due to strict policies and regulations. More research into these implementations can help improve the industry's per-

ception of machine learning and quicken the adoption of these approaches within the financial industry. Current forecasting models, such as GARCH, are widely used for modeling volatility but often fail to capture the intricate behaviors of financial markets that more sophisticated models, such as neural networks, may be able to capture. Our research proposes alternative models, such as neural networks and random forests, to capture the subtle market dynamics that conventional methods sometimes overlook, potentially changing how we make financial predictions. Although the original paper by Lee and Kang (2024) focuses on distributing data over local nodes due to data privacy and security concerns, specialised distribution of training data over individual nodes can have other potential benefits such as specialised nodes that are trained over specific regimes. By clustering financial data, we hope to capture regime shifts and changes in market behaviour such that each node acts as a predictor for a specific regime. We hope that this specialized form of ensemble can provide more nuanced predictions compared to a general model trained over the whole dataset.

The implications of this research extend throughout the financial industry, where time-series data are common, impacting sectors involved in economic forecasting and financial decision-making. By providing alternative modelling techniques to financial institutions, this study is of high interest to institutions, such as banks and hedge funds, to add to their list of models for a more robust financial system. Improved models can lead to better hedging strategies, optimized asset allocations, and more informed policy decisions, thereby contributing to a more secure and reliable financial environment.

Although this research contributes to the application of advanced regression techniques in finance, a field dominated by traditional statistical methods, this research is also relevant for other fields. Specifically, the improvements in methodology such as the usage of random forests and synthetic data integration can also be applied to other unrelated fields to enhance the predictive accuracy of the models.

The contributions of this study include:

- Adapting the methodology proposed by Lee and Kang (2024) for compatibility with random forests, using the spread of tree predictions as a proxy for uncertainty instead of MC-dropout.
- Investigating the effectiveness of this approach on financial time-series datasets, specifically S&P500 daily volatility, comparing its performance to baseline models like GARCH and RiskMetrics.
- Exploring the robustness of the methodology by training local nodes on different data distributions, from simple partitioned data to clustered data.
- Examining the impact of introducing synthetic data to the training set to improve performance in data-scarce nodes, addressing a significant problem noted in Lee and Kang (2024).

The structure of the rest of this thesis is as follows. In Section 2, we compare the existing literature surrounding our research topic and analyse the current approaches used in popular literature. Section 3 explains the research methodology, in which we discuss how we extend

upon the original approach by Lee and Kang (2024). In Section 4, we explain the experimental settings, discuss the results, and explain any meaningful findings. In Section 5, we conclude the study and present the overall contributions of this research.

2 Literature review

2.1 Volatility models

One of the earliest works in modeling time-varying volatility was the Autoregressive Conditional Heteroskedasticity (ARCH) model by Engle (1982), which assumes that the conditional variance of a time series is a function of past squared observations. This model was later developed by Bollerslev (1986) to create the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model, which incorporates both the lagged values of squared observations and lagged conditional variances into model volatility. This model has played a significant role in financial time series for modeling and forecasting the volatility of financial returns (Bollerslev, Chou & Kroner, 1992). This has been used in multiple fields such as risk management, option pricing, and portfolio optimization (Engle & Patton, 2007). As a result, multiple attempts and extensions have been made to improve the GARCH model. The Exponential GARCH (EGARCH) model allows for the asymmetric effects of shocks on volatility (Nelson, 1991), which has the advantage of capturing the leverage effect, where negative shocks have a larger impact on future volatility than positive shocks. The Threshold GARCH (TGARCH) model, also known as GJR-GARCH, similarly models the impact of positive and negative shocks differently on volatility by incorporating threshold effects.

Recent developments in machine learning research have led to a rise in new volatility models. Kim and Won (2018) combines modern machine learning techniques with traditional GARCH models to investigate whether this would improve the forecasting accuracy for volatility modelling. This paper hopes to contribute to the growing literature towards machine learning applications towards volatility modelling.

2.2 Ensemble neural networks

Lee and Kang (2024) explores the idea of utilizing an ensemble of regression neural networks to derive better forecasts. It raises the issue of deteriorating performance in situations of data scarcity and noise in local datasets.

Ensemble methods combine multiple base models to improve predictions in regression and classification problems (Mendes-Moreira, Soares, Jorge & Sousa, 2012) (Rokach, 2010). These methods are popular and can be categorized as static or dynamic. Static ensemble methods apply the same combination rules to each query instance (Breiman, 2001; Freund & Schapire, 1997), but may not suit neural networks trained on different data distributions. Dynamic ensemble methods, however, adjust the combination rules for each instance by weighting models expected to perform well (Britto et al., 2014; Kolter & Maloof, 2007; Rooney et al., 2004). Few studies have focused on regression problems within this context, which this paper addresses.

2.3 Random forests

Random Forests (Breiman, 2001) have become one of the most popular and powerful machine learning algorithms for both classification and regression tasks due to their excellent performance and robustness. This model builds on earlier work by Breiman (2017), who introduced the idea of classification and regression trees (CART), which involves splitting data into subsets based on the most significant feature at each node. Random forests build multiple decision trees based on bootstrap sampling and feature randomization to enhance the prediction accuracy and reduce overfitting. Applications of random forests are vast, ranging from medical uses such as the classification of gene expression data (Díaz-Uriarte & Alvarez de Andrés, 2006) to financial forecasting in the context of stock market prediction (Khaidem, Saha & Dey, 2016). Due to its popularity, many variations of the random forest have been made to further optimize, such as boosting in extreme gradient boosting (XGBoost) (Chen & Guestrin, 2016) and light gradient boosting (LightGBM) (Ke et al., 2017). In this paper, we provide an extension to this approach of dynamic ensemble but extend it to the context of random forests. For this experiment, regular random forests were used for the highest degree of simplicity and generalization. Therefore, the methodology proposed in this paper can be extended to random forest extensions, such as XGBoost and LightGBM.

2.4 Inference

This study utilises inference for the uncertainty quantification stage. As the model's uncertainty is latent for each query instance, different inferential methods can be used to approximate the uncertainty of prediction for a certain query. In the case of a neural network, we adopt the same method as the original paper (Lee & Kang, 2024), which is MC-dropout. Other notable methods such as variational inference (VI) utilises Bayesian statistics for approximate inference, particularly useful when exact Bayesian inference is computationally infeasible by approximating the posterior distributions of model parameters (Blei, Kucukelbir & McAuliffe, 2017). Another method is Monte Carlo Markov Chain (MCMC) (Godsill, 2001) which is flexible as this method can be applied to almost any statistical model. Possible limitations of this approach however is that it can be computationally expensive and there may be convergence issues. Another method is the Deep Ensemble approach where it involves training multiple independent neural networks and using their collective predictions to estimate predictive uncertainty (Lakshminarayanan, Pritzel & Blundell, 2017), however, this is computationally expensive.

For random forests, a slightly different approach is taken. For simplicity, we utilise the variability across bootstrapped samples (Breiman, 1996). For uncertainty quantification using random forests, we use the individual predictions of all the trees, which we then can get the mean and use the standard deviation as the uncertainty metric.

2.5 Synthetic data generation

Financial time-series data often suffer from a high noise-to-signal ratio due to their inherently stochastic nature. This characteristic can make it challenging for models to extract meaningful patterns without additional support. Augmenting the training set with synthetic data can

potentially enhance model generalization by providing more diverse samples. These additional observations can help the model better discern the underlying trend from the noise, as the increased sample size allows for better averaging out of random fluctuations.

Synthetic data generation involves creating new data instances that mimic the distribution of an original dataset. This approach has evolved greatly over time. Early techniques like bootstrapping, which involves randomly selecting data subsets for model training, has been a popular technique within industry and academia to improve model generalization. When repeated with different subsets and aggregated, this process became known as bagging (Mooney, Duval & Duvall, 1993).

Synthetic data serves multiple purposes, including mitigating noise in training sets and expanding dataset size, thereby enhancing overall training set quality. Connor and Khoshgoftaar (2019) explores various data augmentation techniques, including synthetic data generation, to improve model robustness against noise. In image processing, techniques such as rotations and translations help models generalize better to key features in the presence of noisy inputs (Shorten & Khoshgoftaar, 2019).

Recent decades have seen the emergence of advanced generative models. Generative Adversarial Networks (GANs), introduced by (Goodfellow et al., 2020), employ a generator and discriminator network to create realistic synthetic data. Conditional Tabular GAN (CTGAN) (Xu, Skoularidou, Cuesta-Infante & Veeramachaneni, 2019a) is a notable variant designed specifically for tabular data generation. In medical time series data, specialized GANs such as medGAN, ehrGAN, tableGAN, and PATE-GAN have gained prominence.

Variational Autoencoders (VAEs), introduced by (Kingma, Welling et al., 2019), offer another powerful approach. VAEs utilize probabilistic graphical models and learned approximate inference to efficiently produce synthetic samples (Brophy, Wang, She & Ward, 2023). For tabular data, Tabular VAE (TVAE) has been developed as a specialized version. Other methods, such as the Gaussian Copula, model dependencies between variables to generate synthetic data. For our experiment, we opted to use a Copula GAN model (Xu, Skoularidou, Cuesta-Infante & Veeramachaneni, 2019b) due to their computational efficiency.

By addressing data scarcity and noise issues through the introduction of synthetic data, we aim to overcome the performance limitations observed in the dynamic ensemble approach proposed by (Lee & Kang, 2024). This strategy not only increases the volume of available data but also potentially enhances its quality, leading to more robust and generalizable models.

2.6 Local model training on dataset properties

Ensemble learning methods have proven to be robust and versatile, often outperforming single models by aggregating the predictions from multiple learners. However, the effectiveness of an ensemble can vary greatly depending on how the data is partitioned among the models. This literature review explores how the performance of ensemble methods is influenced by various data partitioning strategies, including regular partitioning, bootstrapped partitioning, clustered partitions, and the use of synthetic data.

Kuncheva and Whitaker (2003) discusses the importance of diversity among the base learners in an ensemble. It showed that higher diversity often correlates with improved ensemble per-

formance and shows that the way data is partitioned can influence the diversity of models. Hastie, Tibshirani, Friedman and Friedman (2009) discusses how the properties of data distribution across partitions affect ensemble performance. It notes that disjoint partitions may result in high variance if the data within each partition is not representative of the entire dataset. Breiman (1996) explains how boostrapped sampling can reduce variance and improve model robustness by introducing randomness and reduces the likelihood of overfitting. Chawla, Bowyer, Hall and Kegelmeyer (2002) provides insights into how generating synthetic data can balance class distributions in training datasets, leading to more robust ensemble performance.

2.7 Data smoothing

Data quality improvement often involves noise reduction techniques such as data smoothing (Velleman, 1980). In the context of regression neural networks, synthetic data can enhance model robustness by providing additional relevant information, enabling better understanding and generalization of the underlying data distribution. Various methods have been employed to reduce noise and improve data quality. (Lee & Kang, 2024) utilized Principal Component Analysis (PCA) to identify orthogonal vectors capturing the most variation in the data, effectively ignoring noise-containing components (Wold, Esbensen & Geladi, 1987). Median filtering, which replaces each data point with the median of neighboring points within a specified window, is effective in removing outliers while preserving data edges (Justusson, 2006). Low-pass filters attenuate high-frequency noise while allowing low-frequency components to pass through, smoothing out rapid fluctuations (Roberts & Roberts, 1978). Smoothing splines offer a balance between data fidelity and curve smoothness, making them particularly useful for capturing underlying trends while reducing noise (Wang, 2011). In the realm of financial time-series data, moving averages are a common and effective smoothing technique (Raudys, Lenčiauskas & Malčius, 2013). This method, which we will employ for our financial dataset, involves calculating the average of a subset of data points over a specified time window, effectively smoothing out short-term fluctuations and highlighting longer-term trends Other more sophistical signal processing techniques are also used in finance such as Kalman filtering (Wells, 2013).

3 Methodology

The methodology section is divided into four main sections. To determine the effectiveness of the proposed approach on financial time-series data, the results were compared with those of baseline volatility models. Further techniques are also used to make the time-series data compatible with neural networks, such as creating lagged features, using differencing techniques to ensure stationarity, and using dimension reduction (PCA) to improve data quality. We also explain the steps taken to generate synthetic data and how they are incorporated into our proposed approach. We also introduce our methodology for adapting the approach from Lee and Kang (2024) to a random forest model.

3.1 Baseline volatility models

3.1.1 Historical volatility

Historical volatility is calculated from the daily returns of an asset, providing a measure of the asset's price variability over a specific period. The following steps outline the process for calculating historical volatility. Gather the daily closing prices of the asset over a specific period. Let these prices be denoted as $P_1, P_2, \ldots, P_N$, where P_t is the closing price on day t and N is the total number of days. The daily log return r_t is computed using the formula:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) \tag{1}$$

where P_t is the closing price on day t and P_{t-1} is the closing price on the previous day. Calculate the mean (μ) of the daily log returns over the period:

$$\mu = \frac{1}{N}\sum_{t=1}^{N} r_t \tag{2}$$

where N is the total number of returns. For this study, we use $N = 30$.

The variance (σ^2) of the daily log returns is given by:

$$\sigma^2 = \frac{1}{N-1}\sum_{t=1}^{N}(r_t - \mu)^2 \tag{3}$$

The standard deviation (σ) of the daily log returns, known as historical volatility, is:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(r_t - \mu)^2} \tag{4}$$

3.1.2 Generalized Autoregressive Conditional Heteroskedastic (GARCH) model

The Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model, proposed by Bollerslev (1986), is used to model time series data with time-varying volatility. In summary, the GARCH(p, q) model captures the dynamic structure of the volatility by incorporating past information into the current conditional variance. The GARCH(p, q) model consists of two main components, being the mean equation and the variance equation. The mean equation models the conditional expectation of the process y_t while the variance equation models the conditional variance σ_t^2 of the error term ϵ_t. The GARCH(p, q) model expresses σ_t^2 as a function of past squared errors and past variances. In its simplest form, these equations can be written as:

$$y_t = \mu + \epsilon_t \tag{5}$$

where $\epsilon_t = \sigma_t z_t$, σ_t^2 is the conditional variance, and $z_t \sim \mathcal{N}(0, 1)$ is an i.i.d. standard normal random variable.

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 \tag{6}$$

where ω is a constant term, α_i are coefficients for the lagged squared errors ϵ_{t-i}^2, and β_j are coefficients for the lagged variances σ_{t-j}^2.

For this paper, we opted to use $p = 1$ and $q = 1$ for our baseline model for which we will compare our proposed approach. As a result, the variance equation for the GARCH(1,1) model is:

$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \tag{7}$$

where ω is a constant term, α_1 is the coefficient for the lagged squared residuals ϵ_{t-1}^2, and β_1 is the coefficient for the lagged conditional variance σ_{t-1}^2.

To ensure the model's validity, the parameters must satisfy the conditions:

$$\omega > 0, \quad \alpha_1 \geq 0, \quad \beta_1 \geq 0, \quad \text{and} \quad \alpha_1 + \beta_1 < 1.$$

3.1.3 RiskMetrics

Another popular method used in industry is RiskMetrics, developed by JP Morgan, that weighs observations on an exponentially-weighted basis. The conditional variance σ_t^2 at time t is estimated using the following Exponentially-Weighted Moving Averagage (EWMA) model:

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1 - \lambda) r_{t-1}^2 \tag{8}$$

where λ is the smoothing parameter (decay factor), typically set to 0.94 for daily financial data, r_{t-1} is the return at time $t - 1$, and σ_{t-1}^2 is the estimated variance at time $t - 1$.

The parameter λ controls the rate at which the weights decrease for older observations, with higher values placing more weight on recent returns. Empirical studies have shown that the optimal decay rate across all asset classes turns out to be 0.94 (Zumbach, 2007). As a result, we have set the decay rate $\lambda = 0.94$.

3.2 Synthetic data generation using Gaussian copulas

To help improve the model generalization, particularly in cases with limited or noisy data, which specifically is a significant challenge in financial time-series analysis, we look into the idea of integrating synthetic data into the training set. This study employs a Gaussian Copula model (Patki, Wedge & Veeramachaneni, 2016) to model our data distribution, chosen for its theoretical simplicity compared to more advanced machine learning models such as generative adversarial networks and autoencoders, as well as its computational practicality. The proposed approach establishes a baseline performance that can potentially be achieved by integrating synthetic data into the training set. Further performance improvements may be possible through hyperparameter tuning and the selection of more advanced models for synthetic data generation.

3.2.1 Theoretical background

A Gaussian copula is a type of copula that uses a multivariate normal distribution to describe the dependency structure between variables. The copula separates the dependency structure from the marginal distributions of the variables.

Given a random vector $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with marginal cumulative distribution functions (CDFs) $F_1, F_2, \ldots, F_n$, copula C captures the joint CDF H of $\mathbf{X}$ as follows:

$$H(x_1, x_2, \ldots, x_n) = C(F_1(x_1), F_2(x_2), \ldots, F_n(x_n))$$

For a Gaussian copula, C is derived from the multivariate normal distribution. If Φ_Σ is the CDF of a multivariate normal distribution with zero mean and covariance matrix Σ, the Gaussian copula is defined as:

$$C(u_1, u_2, \ldots, u_n) = \Phi_\Sigma(\Phi^{-1}(u_1), \Phi^{-1}(u_2), \ldots, \Phi^{-1}(u_n))$$

where Φ^{-1} denotes the inverse of the standard normal CDF.

3.2.2 Generating synthetic data

Let $X = (X_1, \ldots, X_d)$ be our random vector of d dimensions that represents the dataset that we are modelling. We model each feature as a univariate normally distributed variable. For each feature $n \in 1, \ldots, d$, we calculate the mean μ_n and variance σ_n^2:

$$\mu_n = \frac{1}{m} \sum_{i=1}^{m} x_{ni}$$

$$\sigma_n^2 = \frac{1}{m-1} \sum_{i=1}^{m} (x_{ni} - \mu_n)^2$$

where m is the number of samples in the dataset.

We then compute the correlation matrix Σ based on the original data and draw a random vector using a multivariate normal sampling function, taking into account the means and the covariance matrix. After the synthetic data are generated, they get integrated with the original training data to enhance the model training.

3.3 Data preparation

3.3.1 Moving average

For the case of the time-series financial data, we do additional data pre-processing to improve model performance. Due to the stochastic nature of the data, we utilise a simple moving average of the time-series data, with a fixed window m set to 10. We choose $m = 10$ to represent an average two trading weeks to give a sufficiently large window size, while still being able to capture weekly changes in volatility.

$$y_i = \frac{1}{m} \sum_{j=i-m}^{i} x_j$$

with y_i being the processed time-series data, x_j being the j-th point in the time series, and m being the window that is taken into account for the moving average.

3.4 Inference (uncertainty quantification)

Statistical inference is an integral intermediary step we take during each query instance to calculate the uncertainty of a certain node's confidence. This is done by quantifying the standard deviation of the query prediction, which can vary in methods based on the context of the problem and model architecture. For different model architectures, different methods of inference are used to support the specific ways that the models function. A description of the inference methods used for both the neural network and random forest are mentioned below.

3.4.1 Neural network

For neural networks, we use MC-dropout for inference. This involves performing multiple forward passes through the network with dropout applied during inference. Lee and Kang (2024) contains detailed explanation of how MC-dropout is implemented for this approach.

3.4.2 Random forest

For Random Forests, each tree T_j in the forest $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ provides a prediction $T_j(\mathbf{x})$. The ensemble prediction for the Random Forest is the average of the predictions from all trees:

$$\hat{y}_{RF} = \frac{1}{m} \sum_{j=1}^{m} T_j(\mathbf{x})$$

with m denoting the number of trees in the random forest.

To use predictive uncertainty in this context, we can define the uncertainty as the spread (variance) of the predictions from the individual trees:

$$u(\mathbf{x}; \mathcal{T}) = \frac{1}{m} \sum_{j=1}^{m} \left(T_j(\mathbf{x}) - \hat{y}_{RF}\right)^2$$

3.5 Weighing predictions

Several methods are employed for weighing predictions such as using the mean and median of all the predictions as a baseline prediction. We assign smaller weights for nodes that has higher predicted uncertainty. Complete description and notation of the different weighing methods can be found in Lee and Kang (2024).

- Oracle
- Mean
- Median
- D-SEL(u) (Dynamic selection unnormalized)
- D-SEL(u') (Dynamic selection normalized)
- D-ENS(u) (Dynamic ensemble unnormalized)
- Proposed method (Dynamic ensemble normalized)

4 Computational experiments

4.1 Datasets used

For this study, we use all the original datasets from Lee and Kang (2024) to compare the methodology extensions. These datasets (Bikesharing, Compactiv, Cpusmall, Ctscan, Indoorloc, Mv, Pole, Puma32h, Telemonitoring) were retrieved from the UCI Machine Learning Repository (Dua & Graff, 2017). These datasets span a whole range of domains from computer systems to telecommunications. By introducing a diverse set of datasets, we hope to see a holistic view of each approach's effectiveness and versatility in different conditions. We also introduce a new dataset of daily volatility from the S&P 500 for the period of January 1 1983 to January 1 2023[1], which is the dataset that we will be mostly focusing on in this study. Information about how this dataset is pre-processed is mentioned in Section 4.2.3. The remaining nine datasets serves to give further insights into the versatility of the proposed approach on different datasets and domains. A description of each dataset used in this study is shown in Table 1.

Table 1: Description of datasets used in this study

Dataset	No. instances	No. features	Domain	Nodes	No. local nodes
Bikesharing	17,379	12	Bike Sharing System	9	[2331, 1379, 1831, 2308, 2201, 1604, 1869, 1816, 1540]
Compactiv	8,192	21	Computer System	4	[4649, 434, 1379, 802]
Cpusmall	8,192	12	Computer System	4	[1904, 1230, 2384, 1204]
Ctscan	53,500	385	Computed Tomography	8	[5124, 8204, 3229, 5456, 4367, 7714, 3449, 13009]
Indoorloc	19,937	585	Indoor Positioning System	7	[2138, 3163, 1226, 2142, 4577, 2014, 2412]
Mv	40,768	10	Synthetic	10	[6205, 3926, 4876, 4204, 3719, 3879, 3604, 3824, 3004, 3527]
Pole	14,998	26	Telecommunication	6	[2404, 2556, 2622, 2777, 1560, 1669]
Puma32h	8,192	32	Robot Arm	10	[803, 832, 764, 760, 814, 843, 802, 843, 849, 882]
Telemonitoring	5,875	20	Parkinson's Disease	7	[562, 1046, 891, 580, 715, 547, 1028]
Volatility	10,053	60	Financial Time-Series	5	[1929, 2038, 1664, 2180, 2242]

4.2 Experimental Settings

This study compares the generalization performance of each model based on the root mean squared error (RMSE) and mean absolute error (MAE). We employed the same uncertainty normalization and weight calculation methodology as proposed by Lee and Kang (2024), which is based on the predicted uncertainty calculated during the inference stage in the query instance for each node.

4.2.1 Neural network settings

We implemented our neural networks using Python 3.9 with the PyTorch library. The architecture and training routine remained consistent with those described in Lee and Kang (2024), with one notable exception. For time series data, we modified the test and train split to enable comparison with baseline models such as GARCH(1,1), which require sequential data training. We maintained an 80/20 split but designated the first 80% of the sample as the training set and the latter 20% as the testing set.

[1]Historical data of the S&P 500 (ĜSPC) was retrieved using the Yahoo finance API, which 40 years (January 1 1983 - January 2023) of historical data was used

4.2.2 Random forest settings

Our random forest implementation used 100 trees and a fixed seed to ensure replicability across iterations and datasets. To prevent divide-by-zero errors, we added a small epsilon value to the standard errors list, as some random forests produced uncertainty values of zero.

4.2.3 Financial data pre-processing

This pre-processing step is unique only for the 40 years of historical volatility data due to the univariate time-series nature of the data. Therefore, extra features would need to be created to be compatible with the dynamic ensemble models. The pre-processing steps varied depending on the model type. For both baseline and ensemble models, we calculated daily returns using the natural logarithm of price differences. For specifically ensemble methods, we applied a 10-day moving average (MA) to the data due to the high noise-to-signal ratio in raw financial data. For feature creation, we added lagged price and price differences for 30 lags, which principal component analysis (PCA) was then used to capture only the 10 largest components with the highest volatility. This approach significantly improved model performance, as shown in Table 8. We assumed zero mean daily returns, equating daily volatility to the absolute value of daily returns. We used historical volatility as our proxy, noting that volatility can be measured through various proxies such as historical and realized volatility.

4.2.4 Synthetic data generation settings

We generated synthetic data for nodes with insufficient data points, defined as those containing fewer data points than the average across all nodes. For these nodes, we generated additional samples to reach the average number of data points per node. To preserve data privacy and security, we trained the data generation models using only the data within each respective node. We utilized the Synthetic Data Vault (SDV) library in Python, specifically employing the Gaussian copula synthesizer for data generation.

4.3 Dataset performance comparisons

We first explore the performance of this approach based on different characteristics of the training set of each node. By default, the original paper opted for a K-Means clustering approach to separate the nodes, but we also explore whether this is a necessary case or whether this actually would perform worse. As mentioned in Lee and Kang (2024), the performance of the approach is likely to perform better when the data distribution within the nodes represent the distribution throughout the whole dataset.

4.3.1 Disjoint Partitioning (DP)

Regular partitioning, also known as disjoint partitioning, involves dividing the dataset into distinct, non-overlapping subsets. Each node receives a unique subset of the data with no overlap with other nodes. This method is straightforward but can result in uneven distributions of data characteristics across nodes, potentially affecting model performance. We implement this by splitting the dataset $\mathbf{X}$ into N disjoint subsets $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N\}$. Then we assign each

subset $\mathbf{X}_i$ to a different node for training local models. For the sake of comparison, we set the number of nodes equal to the number of nodes used in the K-Means Clustering method after filtering out the insufficient nodes.

4.3.2 Bootstrapped Partitioning (BP)

Bootstrapped partitioning involves creating partitions by sampling the dataset with replacement. Each node receives a bootstrapped sample, which may contain duplicate observations. This approach ensures that each node gets a slightly different view of the data, helping to reduce overfitting and improve model robustness. We implement this in the algorithm by sampling d data points with replacement from the dataset $\mathbf{X}$ to form the bootstrapped dataset $\mathbf{X}_i^*$ for each node. For this, d denotes the size of the whole training set divided by the number of nodes so the number of data points trained at each node is identical, similar to disjoint partitioning. Similarly to regular partitioning, the number of nodes is also set to the number of nodes of K-Means clustering method after filtering. By bootstrapping, we hoped that this would enhance model robustness and generalization by benefiting from a diverse set of training data.

4.3.3 K-Means Clustering (KMeans)

K-means clustered partitioning groups the data points into clusters based on their feature similarities using the K-means algorithm. This training approach is what was used in the original paper by Lee and Kang (2024), so we used this as our baseline to directly compare the effects of synthetic data integration. Each node is then assigned data from different clusters. This method ensures that each node has data points that are similar to each other, which can be beneficial for learning local patterns. This results in data that are much more homogeneous within nodes, and therefore reduces intra-node variance but can introduce bias if clusters do not represent the overall distribution. A potential advantage of this approach is that nodes receive data from distinct clusters, which may lead to specialized models. This is implemented by applying K-means clustering to dataset $\mathbf{X}$ to create K clusters $\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_K\}$. Each data point is then labelled with a specific cluster, and the data points are assigned to their respective clusters. For clusters that contain 5% or less data points from the entire dataset, this cluster is ignored in the experiments.

4.3.4 Augmented K-Means Clustering (A-KMeans)

This approach extends the K-means clustered partitioning by adding synthetic data to each cluster. Synthetic data were generated to augment the clusters, particularly for clusters with limited data points. This method helps address data scarcity and improves model training by providing more varied training samples. A potential advantage of this method is that it reduces the variance and potential overfitting by providing additional training data that represent the underlying data distribution. This approach addresses data scarcity and potentially enhances the model performance. One caveat is that this approach relies heavily on the quality of synthetic data, as this can impact model performance negatively, and it is also computationally expensive.

We implement this by firstly performing K-means clustering to divide the dataset $\mathbf{X}$ into clusters $\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_K\}$. Then for each cluster, we generate synthetic data points and ensure

at the end, the training set for each node has at least the number of points that is equal to the size of the whole training set divided by the number of clusters used. Lastly, we combine the original and synthetic data for each cluster and assign them to their respective nodes.

4.4 Methodology comparison results

The format of which the results are showed are shown as: $mean \pm std.dev$. Only the notable results that this paper contributes is listed in the body of the paper. For more complete information about the experimental results, these results are listed in Appendix A.

4.4.1 Random forest methodology results

	RMSE		MAE	
Clustering method	Neural network	Random forest	Neural network	Random forest
DP	2.60 ± 0.80	**1.80 ± 0.87**	2.20 ± 1.25	**1.50 ± 0.81**
BP	**1.50 ± 0.81**	2.00 ± 1.00	**2.10 ± 0.70**	2.20 ± 0.98
KMeans	**1.30 ± 0.64**	2.00 ± 1.10	**2.00 ± 0.63**	2.60 ± 0.92
A-KMeans	1.70 ± 0.90	**1.40 ± 0.80**	2.20 ± 0.75	**2.00 ± 0.89**

Table 2: Average rank comparison

Table 2 presents the average rank values based on their respective models and evaluates the performance of each method across all clustering approaches. The average rank refers to how effective the dynamic ensemble approach is compared to the other baseline weighing methods, as mentioned in Section 3.5. The lower the rank is, the more effective the proposed approach is. This comparison aims to assess whether the dynamic ensemble method is equally effective for the random forest (RF) model compared to the neural network (NN). For each of the error metrics, RMSE and MAE, the superior results are highlighted in bold.

The results reveal that for the dynamic partitioned (DP) and augmented K-Means clustered (A-KMeans) training sets, the random forest tends to achieve a better average rank compared to neural networks. However, it's important to note that this difference is not statistically significant for a given $\alpha = 0.05$. Conversely, the neural network demonstrates greater effectiveness for the bootstrapped partitioned (BP) and K-Means clustered partitions (KMeans), though again, this difference is minimal. These observations suggest that the effectiveness of the dynamic ensemble method is comparable between these two model architectures.

An interesting finding emerges when comparing the results between the K-Means clustered (KMeans) and augmented K-Means clustered (A-KMeans) training sets. The random forest shows to be not as effective for the KMeans case but shows a noticeable increase in average rank The random forest initially shows less effective performance with this set. However, when integrated with synthetic data (A-KMeans), its performance notably improves. Interestingly, the opposite effect is observed for neural networks, where the integration of synthetic data leads to a slight decrease in effectiveness. This could be attributed to the simplistic nature of the random forest which allows for it to generalize better to the underlying trend. While for the neural network, it can be susceptible to overfit to the noise added in the synthetic data generation process, which could worsen overall performance.

In conclusion, the effectiveness of the dynamic ensemble methodology appears to be similar between neural networks and random forests, with only negligible differences observed. The results suggest that both model architectures can be viable options for implementing the dynamic ensemble approach. However, the results reveal a potential advantage for random forests when working with synthetically generated data. This finding indicates that our approach is not limited to neural networks but can be generalized to various model architectures. This adaptability underscores the robustness and versatility of the weighted dynamic ensemble approach in improving predictive accuracy across diverse datasets.

4.4.2 Clustering comparison results

Dataset	DP	BP	KMeans	A-KMeans
Bikesharing	0.7545 ± 0.0833	0.4598 ± 0.0483	0.7493 ± 0.0330	0.7619 ± 0.0820
Compactiv	0.1894 ± 0.0048	0.1156 ± 0.0041	0.1894 ± 0.0056	0.2473 ± 0.0198
Cpusmall	0.1978 ± 0.0049	0.2289 ± 0.0054	0.7332 ± 0.0189	0.7336 ± 0.0686
Ctscan	0.3335 ± 0.0054	0.1265 ± 0.0143	0.3377 ± 0.0299	0.3261 ± 0.0233
Indoorloc	0.1129 ± 0.0014	0.0725 ± 0.0027	0.1402 ± 0.0143	0.1619 ± 0.0249
Mv	0.0249 ± 0.0009	0.0500 ± 0.0036	0.0533 ± 0.0058	0.1979 ± 0.0484
Pole	0.1792 ± 0.0031	0.1182 ± 0.0130	0.3038 ± 0.0316	0.3302 ± 0.0445
Puma32h	0.9598 ± 0.0034	0.7791 ± 0.0027	0.9884 ± 0.0023	0.9900 ± 0.0025
Telemonitoring	0.9733 ± 0.0281	0.5011 ± 0.0246	0.7650 ± 0.0494	0.9016 ± 0.4904
Volatility	0.5741 ± 0.0089	0.5667 ± 0.0174	0.7503 ± 0.0145	0.5791 ± 0.0125
Average rank	2.60 ± 0.80	1.50 ± 0.81	1.30 ± 0.64	1.70 ± 0.90

Table 3: RMSE comparison of neural network trained on different dataset configurations

Dataset	DP	BP	KMeans	A-KMeans
Bikesharing	0.4548 ± 0.0465	0.4658 ± 0.0363	0.4387 ± 0.0246	0.4431 ± 0.0474
Compactiv	0.1226 ± 0.0022	0.1150 ± 0.0028	0.1159 ± 0.0038	0.1404 ± 0.0128
Cpusmall	0.1396 ± 0.0021	0.2250 ± 0.0039	0.2298 ± 0.0031	0.2346 ± 0.0096
Ctscan	0.1217 ± 0.0021	0.1278 ± 0.0169	0.1227 ± 0.0136	0.1243 ± 0.0078
Indoorloc	0.0759 ± 0.0012	0.0737 ± 0.0071	0.0694 ± 0.0034	0.0819 ± 0.0084
Mv	0.0186 ± 0.0006	0.0544 ± 0.0075	0.0533 ± 0.0058	0.0998 ± 0.0235
Pole	0.0881 ± 0.0030	0.1229 ± 0.0160	0.1203 ± 0.0117	0.1319 ± 0.0164
Puma32h	0.7617 ± 0.0027	0.7770 ± 0.0018	0.7778 ± 0.0028	0.7821 ± 0.0033
Telemonitoring	0.6898 ± 0.0263	0.5101 ± 0.0179	0.4852 ± 0.0373	0.6099 ± 0.3475
Volatility	0.3761 ± 0.0088	0.3682 ± 0.0077	0.4678 ± 0.0134	0.3764 ± 0.0060
Average rank	2.20 ± 1.25	2.10 ± 0.70	2.00 ± 0.63	2.20 ± 0.75

Table 4: MAE comparison of neural network trained on different dataset configurations

As shown in Table 3, we can see differences in the effectiveness of the approach based on training the data on clustered nodes or randomly partitioned nodes. For some datasets like Cpusmall, the difference is quite a contrast with regular partitioning having better performance of 0.1978 compared to K-Means clustered data of 0.7332. It seems that for all datasets, excluding Telemonitoring, regular partitioning performs an equally good or better job than clustered data. This could be explained by the nodes being more representative of the overall dataset, and each node being more similar in distribution properties, making more nodes to be accurately

predicted. On the other hand, Table 4, tells us another story. For example, although it seems that training on bootstrap partitioned data for the Bikesharing dataset gives us much better performance in terms of RMSE, when comparing MAE, these differences are nearly negligible. Overall, when comparing the synthetic data integrated approach with the other baseline dataset configurations in the neural network case, it seems to provide the worst performance for both RMSE and MAE metrics.

Dataset	DP	BP	KMeans	A-KMeans
Bikesharing	0.7340 ± 0.0000	0.7117 ± 0.0000	0.7222 ± 0.0000	0.7231 ± 0.0536
Compactiv	0.2325 ± 0.0000	0.3117 ± 0.0000	0.3226 ± 0.0000	0.2996 ± 0.0290
Cpusmall	0.2992 ± 0.0000	0.3211 ± 0.0000	0.3500 ± 0.0000	0.3324 ± 0.0102
Ctscan	0.2997 ± 0.0000	0.3031 ± 0.0000	0.2982 ± 0.0000	0.4485 ± 0.2154
Indoorloc	0.1132 ± 0.0000	0.2866 ± 0.0000	0.2603 ± 0.0000	0.4534 ± 0.1663
Mv	0.0532 ± 0.0000	0.6575 ± 0.0000	0.6415 ± 0.0000	0.4320 ± 0.0917
Pole	0.2845 ± 0.0000	0.3025 ± 0.0000	0.3039 ± 0.0000	0.3030 ± 0.0033
Puma32h	0.9672 ± 0.0000	0.9939 ± 0.0000	0.9947 ± 0.0000	0.9935 ± 0.0017
Telemonitoring	0.9547 ± 0.0000	1.0264 ± 0.0000	1.1443 ± 0.0000	0.7749 ± 0.1097
Volatility	0.7135 ± 0.0000	0.7476 ± 0.0000	0.6780 ± 0.0000	0.6319 ± 0.0000
Average rank	1.80 ± 0.87	2.00 ± 1.00	2.00 ± 1.10	1.40 ± 0.80

Table 5: RMSE comparison of random forest trained on different dataset configurations

Dataset	DP	BP	KMeans	A-KMeans
Bikesharing	0.4701 ± 0.0000	0.4615 ± 0.0000	0.4706 ± 0.0000	0.4730 ± 0.0274
Compactiv	0.1257 ± 0.0000	0.2070 ± 0.0000	0.2165 ± 0.0000	0.1774 ± 0.0204
Cpusmall	0.1438 ± 0.0000	0.1369 ± 0.0000	0.1401 ± 0.0000	0.1375 ± 0.0016
Ctscan	0.0847 ± 0.0000	0.1009 ± 0.0000	0.1008 ± 0.0000	0.1901 ± 0.1492
Indoorloc	0.0577 ± 0.0000	0.1459 ± 0.0000	0.1319 ± 0.0000	0.2030 ± 0.0739
Mv	0.0365 ± 0.0000	0.3308 ± 0.0000	0.3168 ± 0.0000	0.1858 ± 0.0471
Pole	0.1100 ± 0.0000	0.1160 ± 0.0000	0.1186 ± 0.0000	0.1175 ± 0.0018
Puma32h	0.7646 ± 0.0000	0.7838 ± 0.0000	0.7841 ± 0.0000	0.7827 ± 0.0014
Telemonitoring	0.5724 ± 0.0000	0.6371 ± 0.0000	0.7344 ± 0.0000	0.4727 ± 0.0669
Volatility	0.3363 ± 0.0000	0.4161 ± 0.0000	0.4081 ± 0.0000	0.3357 ± 0.0000
Average rank	1.50 ± 0.81	2.20 ± 0.98	2.60 ± 0.92	2.00 ± 0.89

Table 6: MAE comparison of random forest trained on different dataset configurations

Tables 5 and 6 present the results of our random forest implementation. Notably, the standard errors for random forests are insignificant, likely due to their inherent robustness to data noise and less complex nature compared to neural networks. In contrast, neural networks, with their high flexibility, require substantial data to converge to an optimal state that accurately captures the true data distribution. This flexibility results in larger standard errors between experiments for neural networks. Our clustering approach with synthetic data shows varying results across datasets. In some cases, it maintains similar accuracy levels to the baseline K-means configuration, while in others, such as the Mv and Telemonitoring datasets, we observe substantial performance improvements.

Comparing the synthetic data integrated models with the regular K-means clustered models reveals interesting patterns. For random forests, the synthetic data approach outperforms the

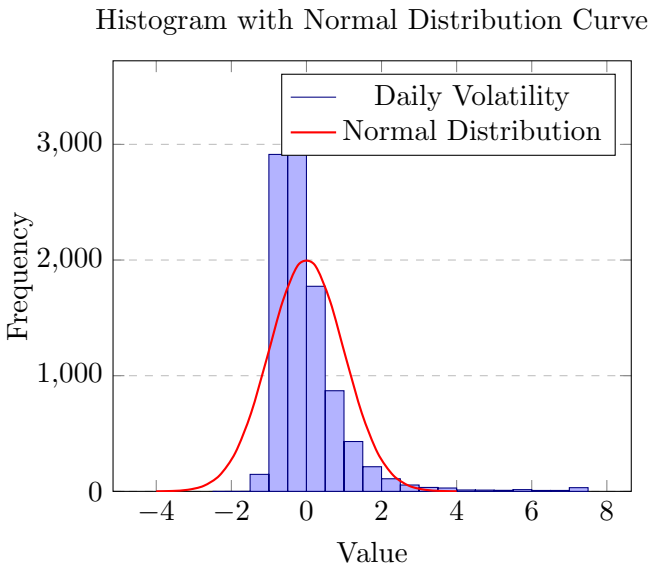

Figure 1: Daily Volatility Distribution Compared to Normal Distribution

regular clustering in seven out of ten datasets, both in terms of RMSE and MAE. However, this trend does not hold for neural networks, where the synthetic data results consistently underperform across all datasets. This contrasting behavior is further evidenced by the average rank comparisons. In the random forest case, the synthetic data approach ranks second only to the regular partitioning configuration, outperforming the regular K-Means dataset configuration. For neural networks, however, the synthetic data approach performs poorly across the board. These findings suggest that simpler model architectures, such as random forests, can benefit more from the integration of synthetic data, likely due to improved generalization. Conversely, the flexibility of neural networks may lead to overfitting on synthetic data, potentially degrading model performance. This underscores the importance of considering model architecture when implementing synthetic data strategies in ensemble learning approaches.

4.5 Time series model comparison results

Table 7: Summary statistics of daily volatility

Mean	**Standard deviation**	**Skewness**	**Kurtosis**	**Jarque-Bera test**
0.75	0.52	4.06	28.02	361840.66

Figure 1 shows the histogram of daily volatility and compares it to the normal distribution. This comparison is necessary as synthetic data generation relies upon the assumption that the features are Gaussian distributed, as it models the data as such. This is important because if the assumption is not met, then the synthetic data would not be representative of the underlying data distribution, therefore making it less effective in improving generalization. This is especially an issue for financial data as it typically has fat tails.

This non-normality is shown in Table 7 where the kurtosis is 28.02, which is greater than 3 (the kurtosis of a normal distribution). A high positive skewness is also shown, being 4.06, providing further evidence against normality. To further assess whether the data is normal, we

performed a Jarque-Bera test. This test evaluates the null hypothesis that the sample data have the skewness and kurtosis matching a normal distribution. Our test yielded a statistic of 361840.66, which far exceeds the critical value of 5.99 for p=0.05. Consequently, we reject the null hypothesis of normality with high confidence. Therefore, we cannot claim that the finance dataset is multivariate normally distributed, so we should consider the synthetic data results with caution.

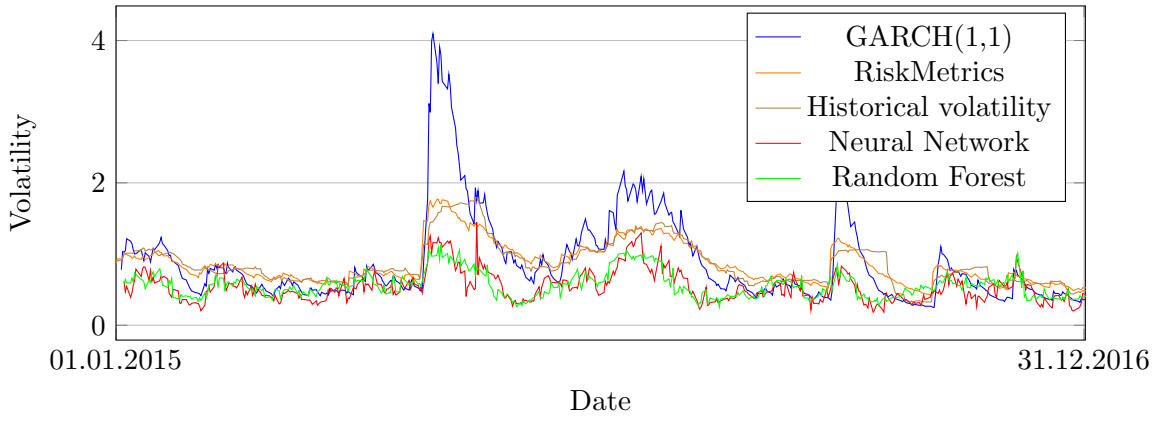

Figure 2: Volatility model comparison chart

Figure 2 shows that ensemble methods perform well during stable periods but underestimate volatility spikes. The ensemble model (neural network) results, trained on A-KMeans data, are shown for direct comparison with baseline methods. Despite the predicted magnitude being far off, ensemble methods accurately capture general volatility trends, with daily movement polarity similar to GARCH(1,1). Historical volatility captures overall levels but lacks directional nuance, often remaining high after spikes. RiskMetrics improves upon this by using exponentially weighted averaging, prioritizing recent information. Unlike historical volatility, which considers a limited time window, GARCH(1,1) and RiskMetrics incorporate information from infinitely previous time frames through error relationships. The dynamic ensemble approach shows promise for capturing relative volatility changes rather than absolute levels.

The random forest case is perhaps the most interesting case of all the models, where it predicts around the same level of volatility at 0.84 and likes to do large spikes during periods of volatility where volatility is greatly different from the usual.

Dataset	RMSE	MAE
Historical volatility	1.6646	1.1149
GARCH(1,1)	3.5571	1.4962
RiskMetrics	1.6687	1.1080
Neural network ensemble (w/o MA smoothing)	1.3397	0.8582
Neural network ensemble (w/ MA smoothing)	1.3376	0.8605
Random forest ensemble (w/o MA smoothing)	1.5647	1.0516
Random forest ensemble (w/ MA smoothing)	1.4271	0.9922

Table 8: Volatility model comparison

Table 8 compares the proposed and baseline models using RMSE and MAE metrics for the period from January 5, 2015, to December 30, 2022. For the neural network and random forest

results, we use the dynamic ensemble models trained on A-KMeans data to show specifically the performance of our proposed method. These outperformed the baseline models in terms of RMSE and MAE. However, as shown in Figure 2, baseline models provide more nuanced forecasts. A comprehensive comparison requires examining the actual predicted values and their practical utility.

The neural network model demonstrates the best performance in both RMSE and MAE. Unexpectedly, the GARCH(1,1) model performed worst overall, with the highest RMSE (3.5571) and MAE (1.4962). Nevertheless, it does well at capturing rapid volatility changes, while other models like historical volatility and neural networks may lag in detecting such fluctuations.

The results also show that training models on smoothened data showed a clear increase in model performance. For the random forest ensemble, moving average (MA) smoothing reduced RMSE by 9% and MAE by 6%. However, the neural network ensemble showed negligible differences between regular and smoothened data in terms of RMSE and MAE.

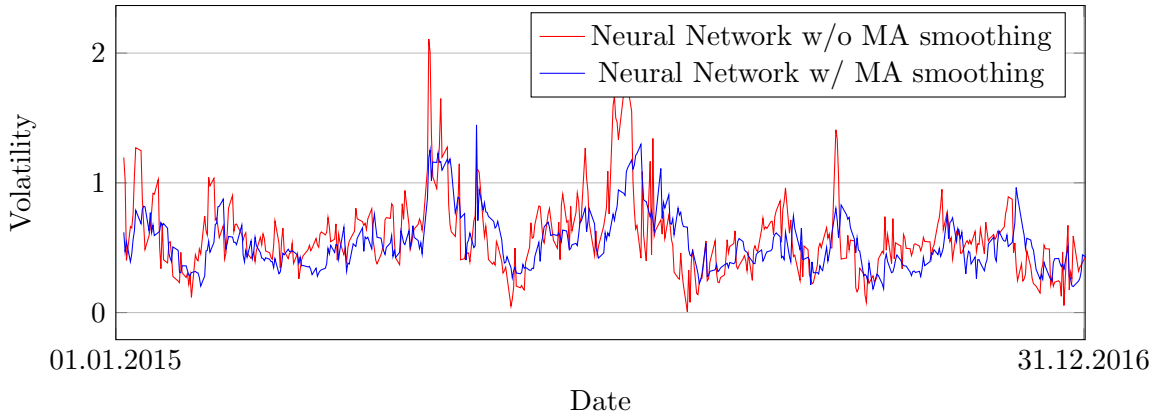

Figure 3: Neural network model comparison between noisy and denoised training sets

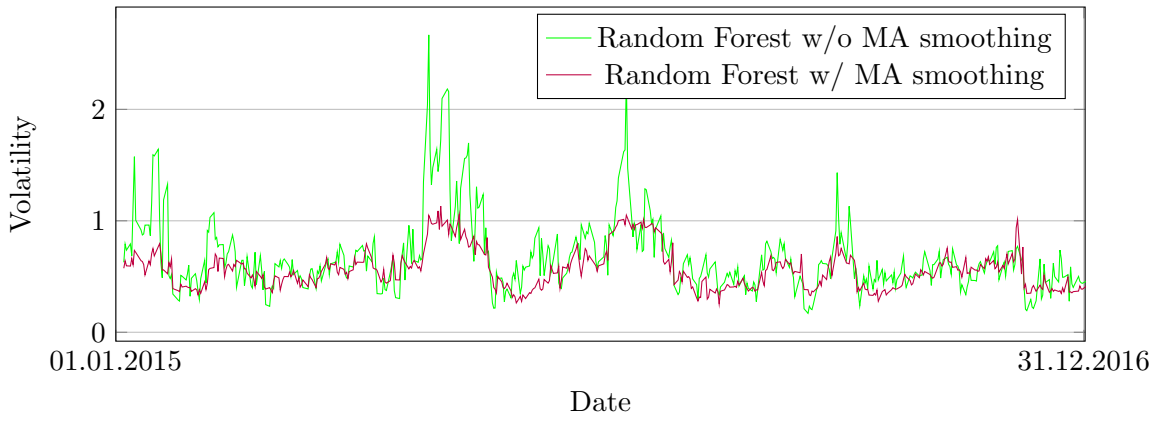

Figure 4: Random forest model comparison between noisy and denoised training sets

Figures 3 and 4 illustrate the contrasts between models trained on regular noisy data and those trained on denoised data. The denoising process involved smoothing the time-series data using a moving average technique. As evidenced by these figures, the models trained on denoised data produce smoother predictions. However, this smoothing comes with a trade-off: sudden spikes in volatility tend to be underestimated. This occurs because the moving average naturally

dampens extreme values, potentially leading to an underestimation of volatility during highly volatile periods. Thus, while the denoised model offers more stable predictions overall, it may not fully capture rapid or extreme market fluctuations.

5 Conclusion

In this study, we proposed methodological improvements to the dynamic ensembling of regression neural networks based on predictive uncertainty, building upon the approach initially proposed by Lee and Kang (2024). When comparing between different data distributions, our findings reveal that this approach is most effective when training data distributions across nodes are similar. When applying this methodology to random forest models, we observed good and more stable performance, likely due to their simpler structure compared to neural networks, which led to a lower risk of overfitting. In the context of time-series financial forecasting, we compared our approach with conventional volatility modeling methods such as GARCH(p,q) and RiskMetrics. In terms of RMSE and MAE metrics, they both overperformed baseline methods, with the neural network having the best overall performance. However, when examining further into the dynamics, the accuracy of these models vary greatly depending on the current market conditions. While ensemble methods struggled during periods of high volatility, this could potentially be mitigated by introducing additional features. A notable advantage of our approach over traditional methods is the ability to incorporate alternative data, such as sentiment analysis, which could enhance predictions.

The introduction of synthetic data showed promising results, particularly in random forest models. However, its implementation in neural networks led to decreased performance, suggesting a potential trade-off between including synthetic data for better generalization and maintaining model flexibility. As a result, the A-KMeans clustering method seems to be the preferable method for random forests, however, for neural networks, the original KMeans clustering approach seems to be preferable. Future research could explore alternative methods of synthetic data generation, such as CTGAN or TVAE, to enhance the quality of training data for regression models. Another interesting direction would be to examine the potential benefits of using specific model architectures for individual nodes, while addressing the challenge of reconciling different uncertainty quantification methods across varied model architectures. Other more sophisticated signal processing techniques such as Kalman filtering could potentially be used in replacement to the moving average approach.

In conclusion, while our proposed improvements show promise, particularly in certain contexts, they also highlight areas requiring further research. The varying performance across different model types and data conditions underscores the complexity of financial forecasting and the need for continued refinement of ensemble methods in this field. As we continue to develop and refine these approaches, we move closer to more accurate and robust financial forecasting models, with potential impacts across various sectors of the financial industry.

References

Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. (2017). Variational inference: A review for

statisticians. *Journal of the American statistical Association*, *112*(518), 859–877.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, *31*(3), 307–327.

Bollerslev, T., Chou, R. Y. & Kroner, K. F. (1992). Arch modeling in finance: A review of the theory and empirical evidence. *Journal of econometrics*, *52*(1-2), 5–59.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*, 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5–32.

Breiman, L. (2017). *Classification and regression trees*. Routledge.

Brophy, E., Wang, Z., She, Q. & Ward, T. (2023). Generative adversarial networks in time series: A systematic literature review. *ACM Computing Surveys*, *55*(10), 1–31.

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Connor, S. & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, *6*(1), 1–48.

Díaz-Uriarte, R. & Alvarez de Andrés, S. (2006). Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, *7*, 1–13.

Dua, D. & Graff, C. (2017). *Uci machine learning repository.* Retrieved from `http://archive.ics.uci.edu/ml`

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society*, 987–1007.

Engle, R. F. & Patton, A. J. (2007). What good is a volatility model? In *Forecasting volatility in the financial markets* (pp. 47–63). Elsevier.

Godsill, S. J. (2001). On the relationship between markov chain monte carlo methods for model uncertainty. *Journal of computational and graphical statistics*, *10*(2), 230–248.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144.

Hastie, T., Tibshirani, R., Friedman, J. H. & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2). Springer.

Justusson, B. (2006). Median filtering: Statistical properties. *Two-dimensional digital signal prcessing II: transforms and median filters*, 161–196.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, *30*.

Khaidem, L., Saha, S. & Dey, S. R. (2016). Predicting the direction of stock market prices using random forest. *arXiv preprint arXiv:1605.00003*.

Kim, H. Y. & Won, C. H. (2018). Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple garch-type models. *Expert Systems with Applications*, *103*, 25–37.

Kingma, D. P., Welling, M. et al. (2019). An introduction to variational autoencoders. *Found-*

ations and Trends® in Machine Learning, *12*(4), 307–392.

Kuncheva, L. I. & Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, *51*, 181–207.

Lakshminarayanan, B., Pritzel, A. & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, *30*.

Lee, Y. & Kang, S. (2024). Dynamic ensemble of regression neural networks based on predictive uncertainty. *Computers & Industrial Engineering*, 110011.

Mendes-Moreira, J., Soares, C., Jorge, A. M. & Sousa, J. F. D. (2012). Ensemble approaches for regression: A survey. *Acm computing surveys (csur)*, *45*(1), 1–40.

Mooney, C. Z., Duval, R. D. & Duvall, R. (1993). *Bootstrapping: A nonparametric approach to statistical inference* (No. 95). sage.

Nelson, D. B. (1991). Conditional heteroskedasticity in asset returns: A new approach. *Econometrica: Journal of the econometric society*, 347–370.

Patki, N., Wedge, R. & Veeramachaneni, K. (2016, Oct). The synthetic data vault. In *Ieee international conference on data science and advanced analytics (dsaa)* (p. 399-410). doi: 10.1109/DSAA.2016.49

Raudys, A., Lenčiauskas, V. & Malčius, E. (2013). Moving averages for financial data smoothing. In *Information and software technologies: 19th international conference, icist 2013, kaunas, lithuania, october 2013. proceedings 19* (pp. 34–45).

Roberts, J. & Roberts, T. D. (1978). Use of the butterworth low-pass filter for oceanographic data. *Journal of Geophysical Research: Oceans*, *83*(C11), 5510–5514.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial intelligence review*, *33*, 1–39.

Shorten, C. & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, *6*(1), 1–48.

Velleman, P. F. (1980). Definition and comparison of robust nonlinear data smoothing algorithms. *Journal of the American Statistical Association*, *75*(371), 609–615.

Wang, Y. (2011). *Smoothing splines: methods and applications*. CRC press.

Wells, C. (2013). *The kalman filter in finance* (Vol. 32). Springer Science & Business Media.

Wold, S., Esbensen, K. & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*(1-3), 37–52.

Xu, L., Skoularidou, M., Cuesta-Infante, A. & Veeramachaneni, K. (2019a). Modeling tabular data using conditional gan. *Advances in neural information processing systems*, *32*.

Xu, L., Skoularidou, M., Cuesta-Infante, A. & Veeramachaneni, K. (2019b). Modeling tabular data using conditional gan. In *Advances in neural information processing systems.*

Zumbach, G. O. (2007). The riskmetrics 2006 methodology. *Available at SSRN 1420185*.

A Full experimental results

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.3862 ± 0.0069	0.6800 ± 0.0194	0.6826 ± 0.0281	0.9338 ± 0.0875	0.8153 ± 0.0805	0.9281 ± 0.0874	0.7545 ± 0.0833
Compactiv	0.2055 ± 0.0103	0.1851 ± 0.0044	0.1824 ± 0.0052	0.2045 ± 0.0109	0.2091 ± 0.0100	0.2006 ± 0.0092	0.1894 ± 0.0048
Cpusmall	0.2033 ± 0.0051	0.1929 ± 0.0028	0.1914 ± 0.0023	0.2178 ± 0.0118	0.2169 ± 0.0115	0.2147 ± 0.0116	0.1978 ± 0.0049
Ctscan	0.0613 ± 0.0026	0.3156 ± 0.0023	0.3251 ± 0.0029	0.3521 ± 0.0090	0.3543 ± 0.0082	0.3505 ± 0.0094	0.3335 ± 0.0054
Indoorloc	0.0959 ± 0.0017	0.2139 ± 0.0170	0.1270 ± 0.0040	0.1209 ± 0.0065	0.1185 ± 0.0017	0.1201 ± 0.0064	0.1129 ± 0.0014
Mv	0.0333 ± 0.0011	0.0194 ± 0.0007	0.0199 ± 0.0007	0.0309 ± 0.0011	0.0328 ± 0.0009	0.0302 ± 0.0011	0.0249 ± 0.0009
Pole	0.1984 ± 0.0050	0.1771 ± 0.0034	0.1749 ± 0.0035	0.1876 ± 0.0057	0.1908 ± 0.0044	0.1855 ± 0.0054	0.1792 ± 0.0031
Puma32	0.9591 ± 0.0021	0.9519 ± 0.0006	0.9522 ± 0.0008	0.9605 ± 0.0021	0.9661 ± 0.0051	0.9602 ± 0.0021	0.9598 ± 0.0034
Telemonitoring	0.2358 ± 0.0108	0.8936 ± 0.0087	0.9341 ± 0.0183	1.0655 ± 0.0497	1.0202 ± 0.0280	1.0620 ± 0.0492	0.9733 ± 0.0281
Volatility	0.5593 ± 0.0105	1.4243 ± 0.1330	0.9106 ± 0.0552	0.6145 ± 0.0190	0.5934 ± 0.0173	0.6106 ± 0.0178	0.5741 ± 0.0089
Average rank		2.30 ± 1.90	2.30 ± 1.42	5.10 ± 0.70	4.70 ± 1.55	4.00 ± 0.63	2.60 ± 0.80

Table 9: RMSE scores of neural network models trained with DP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.2445 ± 0.0041	0.4661 ± 0.0172	0.4308 ± 0.0166	0.5517 ± 0.0549	0.4850 ± 0.0470	0.5485 ± 0.0546	0.4548 ± 0.0465
Compactiv	0.1294 ± 0.0026	0.1209 ± 0.0016	0.1206 ± 0.0019	0.1273 ± 0.0026	0.1286 ± 0.0026	0.1261 ± 0.0022	0.1226 ± 0.0022
Cpusmall	0.1435 ± 0.0023	0.1370 ± 0.0017	0.1364 ± 0.0018	0.1429 ± 0.0025	0.1439 ± 0.0032	0.1420 ± 0.0025	0.1396 ± 0.0021
Ctscan	0.0339 ± 0.0010	0.1589 ± 0.0017	0.1384 ± 0.0014	0.1246 ± 0.0025	0.1260 ± 0.0027	0.1242 ± 0.0026	0.1217 ± 0.0021
Indoorloc	0.0642 ± 0.0008	0.1646 ± 0.0147	0.0881 ± 0.0036	0.0768 ± 0.0022	0.0781 ± 0.0014	0.0766 ± 0.0022	0.0759 ± 0.0012
Mv	0.0253 ± 0.0008	0.0139 ± 0.0005	0.0144 ± 0.0005	0.0235 ± 0.0008	0.0249 ± 0.0007	0.0230 ± 0.0008	0.0186 ± 0.0006
Pole	0.1039 ± 0.0036	0.0925 ± 0.0030	0.0901 ± 0.0030	0.0890 ± 0.0037	0.0931 ± 0.0033	0.0882 ± 0.0037	0.0881 ± 0.0030
Puma32	0.7622 ± 0.0017	0.7561 ± 0.0005	0.7561 ± 0.0009	0.7601 ± 0.0017	0.7669 ± 0.0049	0.7598 ± 0.0016	0.7617 ± 0.0027
Telemonitoring	0.1390 ± 0.0096	0.7315 ± 0.0125	0.6852 ± 0.0149	0.7503 ± 0.0434	0.7043 ± 0.0268	0.7493 ± 0.0432	0.6898 ± 0.0263
Volatility	0.3494 ± 0.0027	1.0331 ± 0.0840	0.6357 ± 0.0307	0.3913 ± 0.0136	0.3845 ± 0.0120	0.3893 ± 0.0137	0.3761 ± 0.0088
Average rank		3.70 ± 1.85	2.60 ± 1.80	4.40 ± 1.11	4.70 ± 1.42	3.40 ± 1.11	2.20 ± 1.25

Table 10: MAE scores of neural network models trained with DP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.2192 ± 0.0064	0.8781 ± 0.0549	0.5906 ± 0.0387	0.5176 ± 0.0913	0.4616 ± 0.0540	0.5151 ± 0.0911	0.4598 ± 0.0483
Compactiv	0.1051 ± 0.0022	0.1592 ± 0.0131	0.1439 ± 0.0113	0.1094 ± 0.0016	0.1184 ± 0.0047	0.1094 ± 0.0016	0.1156 ± 0.0041
Cpusmall	0.1071 ± 0.0011	0.2179 ± 0.0051	0.2424 ± 0.0058	0.2386 ± 0.0052	0.2385 ± 0.0070	0.2380 ± 0.0051	0.2289 ± 0.0054
Ctscan	0.0312 ± 0.0014	0.5591 ± 0.0148	0.4909 ± 0.0164	0.3095 ± 0.0307	0.1147 ± 0.0152	0.3092 ± 0.0308	0.1265 ± 0.0143
Indoorloc	0.0474 ± 0.0025	0.8846 ± 0.0289	0.3659 ± 0.0342	0.0882 ± 0.0111	0.0688 ± 0.0026	0.0886 ± 0.0112	0.0725 ± 0.0027
Mv	0.0149 ± 0.0006	0.5760 ± 0.0260	0.3678 ± 0.0244	0.8547 ± 0.0505	0.0296 ± 0.0034	0.9358 ± 0.0677	0.0500 ± 0.0036
Pole	0.0699 ± 0.0023	0.5368 ± 0.0273	0.2545 ± 0.0135	0.1343 ± 0.0570	0.1234 ± 0.0136	0.1397 ± 0.0550	0.1182 ± 0.0130
Puma32	0.7788 ± 0.0023	0.7723 ± 0.0015	0.7730 ± 0.0018	0.7929 ± 0.0044	0.7883 ± 0.0025	0.7928 ± 0.0042	0.7791 ± 0.0027
Telemonitoring	0.1714 ± 0.0059	0.7216 ± 0.0166	0.6284 ± 0.0218	0.6626 ± 0.0281	0.5057 ± 0.0305	0.6589 ± 0.0273	0.5011 ± 0.0246
Volatility	0.5411 ± 0.0066	0.5598 ± 0.0119	0.5609 ± 0.0141	0.6155 ± 0.0277	0.6153 ± 0.0263	0.6074 ± 0.0282	0.5667 ± 0.0174
Average rank		4.10 ± 1.97	3.70 ± 1.49	4.80 ± 1.17	3.00 ± 1.10	3.90 ± 1.30	1.50 ± 0.81

Table 11: RMSE scores of neural network models trained with BP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.2285 ± 0.0073	0.8362 ± 0.0794	0.5711 ± 0.0375	0.4896 ± 0.0471	0.4740 ± 0.0374	0.4865 ± 0.0468	0.4658 ± 0.0363
Compactiv	0.1029 ± 0.0009	0.1665 ± 0.0070	0.1547 ± 0.0110	0.1086 ± 0.0010	0.1182 ± 0.0031	0.1085 ± 0.0010	0.1150 ± 0.0028
Cpusmall	0.1078 ± 0.0017	0.2186 ± 0.0044	0.2415 ± 0.0023	0.2408 ± 0.0066	0.2366 ± 0.0036	0.2405 ± 0.0066	0.2250 ± 0.0039
Ctscan	0.0318 ± 0.0014	0.5544 ± 0.0141	0.4934 ± 0.0195	0.2915 ± 0.0401	0.1175 ± 0.0175	0.2913 ± 0.0400	0.1278 ± 0.0169
Indoorloc	0.0454 ± 0.0012	0.8614 ± 0.0554	0.3577 ± 0.0233	0.0954 ± 0.0120	0.0695 ± 0.0077	0.0958 ± 0.0124	0.0737 ± 0.0071
Mv	0.0151 ± 0.0006	0.5850 ± 0.0179	0.3613 ± 0.0320	0.8684 ± 0.0702	0.0333 ± 0.0063	0.9535 ± 0.0835	0.0544 ± 0.0075
Pole	0.0700 ± 0.0035	0.5295 ± 0.0257	0.2506 ± 0.0256	0.1214 ± 0.0158	0.1278 ± 0.0166	0.1233 ± 0.0149	0.1229 ± 0.0160
Puma32	0.7771 ± 0.0018	0.7720 ± 0.0008	0.7723 ± 0.0009	0.7973 ± 0.0075	0.7854 ± 0.0032	0.7972 ± 0.0076	0.7770 ± 0.0018
Telemonitoring	0.1777 ± 0.0046	0.7294 ± 0.0171	0.6466 ± 0.0260	0.6740 ± 0.0196	0.5138 ± 0.0218	0.6708 ± 0.0193	0.5101 ± 0.0179
Volatility	0.3578 ± 0.0034	0.3649 ± 0.0063	0.3667 ± 0.0077	0.3865 ± 0.0106	0.3856 ± 0.0096	0.3830 ± 0.0107	0.3682 ± 0.0077
Average rank		4.30 ± 2.24	4.10 ± 1.37	4.10 ± 1.58	2.70 ± 1.42	3.70 ± 1.27	2.10 ± 0.70

Table 12: MAE scores of neural network models trained with BP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.3621 ± 0.0144	1.1409 ± 0.0998	0.8639 ± 0.0582	0.7901 ± 0.0499	0.7890 ± 0.0275	0.7830 ± 0.0517	0.7493 ± 0.0330
Compactiv	0.1526 ± 0.0017	0.2194 ± 0.0113	0.2119 ± 0.0119	0.2132 ± 0.0078	0.1948 ± 0.0183	0.2135 ± 0.0071	0.1894 ± 0.0056
Cpusmall	0.1500 ± 0.0023	0.6267 ± 0.0085	0.7892 ± 0.0140	0.7916 ± 0.0227	0.7789 ± 0.0213	0.7911 ± 0.0223	0.7332 ± 0.0189
Ctscan	0.0570 ± 0.0021	0.7227 ± 0.0158	0.7348 ± 0.0227	0.7329 ± 0.0555	0.3816 ± 0.0322	0.7304 ± 0.0554	0.3377 ± 0.0299
Indoorloc	0.0693 ± 0.0028	0.9968 ± 0.0616	0.5250 ± 0.0533	0.1962 ± 0.0300	0.1569 ± 0.0215	0.1954 ± 0.0297	0.1402 ± 0.0143
Mv	0.0236 ± 0.0012	0.7930 ± 0.0201	0.7704 ± 0.0531	1.3433 ± 0.0565	0.1668 ± 0.0288	1.3761 ± 0.0546	0.1551 ± 0.0188
Pole	0.1480 ± 0.0044	0.6549 ± 0.0198	0.3874 ± 0.0208	0.3228 ± 0.0338	0.3395 ± 0.0379	0.3237 ± 0.0307	0.3038 ± 0.0316
Puma32	0.9848 ± 0.0033	0.9844 ± 0.0015	0.9830 ± 0.0008	1.0188 ± 0.0086	0.9962 ± 0.0029	1.0186 ± 0.0086	0.9884 ± 0.0023
Telemonitoring	0.2816 ± 0.0039	0.9204 ± 0.0209	0.8817 ± 0.0484	1.0305 ± 0.0264	0.8401 ± 0.0520	1.0210 ± 0.0250	0.7650 ± 0.0494
Volatility	0.9712 ± 0.1156	0.7828 ± 0.0300	0.7556 ± 0.0221	1.3320 ± 0.1024	0.8122 ± 0.0193	1.3286 ± 0.1051	0.7503 ± 0.0145
Average rank		4.10 ± 1.76	3.70 ± 1.49	4.80 ± 1.25	2.80 ± 0.87	4.30 ± 1.19	1.30 ± 0.64

Table 13: RMSE scores of neural network models trained with KMeans data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.2283 ± 0.0074	0.8563 ± 0.1069	0.5427 ± 0.0335	0.4662 ± 0.0359	0.4417 ± 0.0253	0.4634 ± 0.0361	0.4387 ± 0.0246
Compactiv	0.1027 ± 0.0010	0.1663 ± 0.0126	0.1555 ± 0.0133	0.1091 ± 0.0015	0.1187 ± 0.0046	0.1091 ± 0.0015	0.1159 ± 0.0038
Cpusmall	0.1079 ± 0.0016	0.2178 ± 0.0034	0.2424 ± 0.0041	0.2426 ± 0.0062	0.2403 ± 0.0040	0.2420 ± 0.0062	0.2298 ± 0.0031
Ctscan	0.0316 ± 0.0006	0.5672 ± 0.0227	0.4937 ± 0.0155	0.3079 ± 0.0360	0.1093 ± 0.0133	0.3078 ± 0.0359	0.1227 ± 0.0136
Indoorloc	0.0459 ± 0.0012	0.8967 ± 0.0506	0.3735 ± 0.0299	0.0831 ± 0.0081	0.0649 ± 0.0037	0.0832 ± 0.0080	0.0694 ± 0.0034
Mv	0.0151 ± 0.0007	0.5821 ± 0.0154	0.3752 ± 0.0320	0.8388 ± 0.0797	0.0323 ± 0.0040	0.9231 ± 0.0843	0.0533 ± 0.0058
Pole	0.0701 ± 0.0024	0.5290 ± 0.0169	0.2458 ± 0.0142	0.1169 ± 0.0114	0.1253 ± 0.0130	0.1203 ± 0.0098	0.1203 ± 0.0117
Puma32	0.7788 ± 0.0017	0.7725 ± 0.0004	0.7725 ± 0.0005	0.7978 ± 0.0067	0.7856 ± 0.0032	0.7976 ± 0.0068	0.7778 ± 0.0028
Telemonitoring	0.1745 ± 0.0042	0.7257 ± 0.0166	0.6284 ± 0.0337	0.6475 ± 0.0222	0.4838 ± 0.0438	0.6442 ± 0.0228	0.4852 ± 0.0373
Volatility	0.6511 ± 0.0812	0.4865 ± 0.0234	0.4795 ± 0.0195	0.7577 ± 0.0813	0.5025 ± 0.0131	0.7559 ± 0.0825	0.4678 ± 0.0134
Average rank		4.50 ± 2.01	4.00 ± 1.26	4.20 ± 1.66	2.50 ± 1.36	3.80 ± 1.33	2.00 ± 0.63

Table 14: MAE scores of neural network models trained with KMeans data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.3816 ± 0.0143	1.1381 ± 0.1024	0.9285 ± 0.1048	0.8918 ± 0.1319	0.8130 ± 0.0913	0.8821 ± 0.1318	0.7619 ± 0.0820
Compactiv	0.1965 ± 0.0117	0.2327 ± 0.0134	0.2322 ± 0.0118	0.2637 ± 0.0286	0.2621 ± 0.0249	0.2627 ± 0.0281	0.2473 ± 0.0198
Cpusmall	0.1558 ± 0.0042	0.6302 ± 0.0344	0.7727 ± 0.0745	0.8326 ± 0.0417	0.7840 ± 0.0673	0.8324 ± 0.0417	0.7336 ± 0.0686
Ctscan	0.0825 ± 0.0042	0.7395 ± 0.0287	0.7705 ± 0.0505	0.5642 ± 0.0418	0.3668 ± 0.0243	0.5606 ± 0.0417	0.3261 ± 0.0233
Indoorloc	0.0725 ± 0.0025	0.9618 ± 0.0395	0.5142 ± 0.0411	0.5753 ± 0.1401	0.1884 ± 0.0354	0.5741 ± 0.1397	0.1619 ± 0.0249
Mv	0.0607 ± 0.0039	0.7016 ± 0.0959	0.5101 ± 0.1536	1.4191 ± 0.0555	0.2303 ± 0.0594	1.4281 ± 0.0588	0.1979 ± 0.0484
Pole	0.1483 ± 0.0025	0.6483 ± 0.0195	0.3957 ± 0.0175	0.3536 ± 0.0458	0.3673 ± 0.0510	0.3487 ± 0.0455	0.3302 ± 0.0445
Puma32	0.9893 ± 0.0021	0.9856 ± 0.0013	0.9854 ± 0.0020	1.0292 ± 0.0198	0.9978 ± 0.0041	1.0289 ± 0.0196	0.9900 ± 0.0025
Telemonitoring	0.5027 ± 0.6232	1.0228 ± 0.4505	0.9687 ± 0.4661	1.1432 ± 0.4163	0.9609 ± 0.4764	1.1354 ± 0.4173	0.9016 ± 0.4904
Volatility	0.5307 ± 0.0045	0.5728 ± 0.0104	0.5731 ± 0.0107	0.6262 ± 0.0242	0.6330 ± 0.0244	0.6174 ± 0.0231	0.5791 ± 0.0125
Average rank		3.70 ± 1.95	3.20 ± 1.60	5.00 ± 1.00	3.20 ± 1.33	4.20 ± 1.17	1.70 ± 0.90

Table 15: RMSE scores of neural network models trained with A-KMeans data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.2417 ± 0.0081	0.8569 ± 0.1018	0.6056 ± 0.0623	0.5146 ± 0.0861	0.4502 ± 0.0517	0.5108 ± 0.0861	0.4431 ± 0.0474
Compactiv	0.1135 ± 0.0014	0.1624 ± 0.0146	0.1488 ± 0.0107	0.1322 ± 0.0173	0.1466 ± 0.0152	0.1319 ± 0.0172	0.1404 ± 0.0128
Cpusmall	0.1123 ± 0.0023	0.2392 ± 0.0141	0.2535 ± 0.0130	0.2722 ± 0.0205	0.2450 ± 0.0105	0.2717 ± 0.0204	0.2346 ± 0.0096
Ctscan	0.0418 ± 0.0014	0.5563 ± 0.0242	0.5255 ± 0.0362	0.1986 ± 0.0192	0.1111 ± 0.0093	0.1984 ± 0.0192	0.1243 ± 0.0078
Indoorloc	0.0499 ± 0.0016	0.8656 ± 0.0340	0.3741 ± 0.0320	0.3007 ± 0.1106	0.0791 ± 0.0095	0.3009 ± 0.1103	0.0819 ± 0.0084
Mv	0.0349 ± 0.0022	0.5594 ± 0.0701	0.3213 ± 0.0791	0.9127 ± 0.0664	0.0909 ± 0.0248	0.9404 ± 0.0777	0.0998 ± 0.0235
Pole	0.0711 ± 0.0022	0.5258 ± 0.0154	0.2537 ± 0.0114	0.1276 ± 0.0141	0.1361 ± 0.0172	0.1266 ± 0.0137	0.1319 ± 0.0164
Puma32	0.7837 ± 0.0015	0.7768 ± 0.0016	0.7771 ± 0.0027	0.8125 ± 0.0187	0.7892 ± 0.0044	0.8122 ± 0.0185	0.7821 ± 0.0033
Telemonitoring	0.3347 ± 0.4386	0.7819 ± 0.2894	0.7000 ± 0.3146	0.7520 ± 0.3014	0.6083 ± 0.3507	0.7489 ± 0.3016	0.6099 ± 0.3475
Volatility	0.3555 ± 0.0016	0.3746 ± 0.0061	0.3759 ± 0.0053	0.3927 ± 0.0082	0.3965 ± 0.0092	0.3893 ± 0.0082	0.3764 ± 0.0060
Average rank		4.40 ± 2.11	3.90 ± 1.22	4.20 ± 1.40	2.70 ± 1.68	3.60 ± 1.56	2.20 ± 0.75

Table 16: MAE scores of neural network models trained with A-KMeans data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.6058 ± 0.0000	0.7927 ± 0.0000	0.7826 ± 0.0000	0.8980 ± 0.0000	0.7700 ± 0.0000	0.8907 ± 0.0000	0.7340 ± 0.0000
Compactiv	0.2948 ± 0.0000	0.2520 ± 0.0000	0.2424 ± 0.0000	0.2551 ± 0.0000	0.2224 ± 0.0000	1.5186 ± 0.0000	0.2325 ± 0.0000
Cpusmall	0.3164 ± 0.0000	0.2833 ± 0.0000	0.2903 ± 0.0000	0.3218 ± 0.0000	0.3187 ± 0.0000	1.5371 ± 0.0000	0.2992 ± 0.0000
Ctscan	0.1250 ± 0.0000	0.3588 ± 0.0000	0.3476 ± 0.0000	0.3280 ± 0.0000	0.3214 ± 0.0000	0.4256 ± 0.0000	0.2997 ± 0.0000
Indoorloc	0.0779 ± 0.0000	0.3523 ± 0.0000	0.1509 ± 0.0000	0.1208 ± 0.0000	0.1341 ± 0.0000	0.2947 ± 0.0000	0.1132 ± 0.0000
Mv	0.0639 ± 0.0000	0.0529 ± 0.0000	0.0525 ± 0.0000	0.0593 ± 0.0000	0.0589 ± 0.0000	0.3850 ± 0.0000	0.0532 ± 0.0000
Pole	0.3053 ± 0.0000	0.2847 ± 0.0000	0.2845 ± 0.0000	0.3027 ± 0.0000	0.3039 ± 0.0000	3.0038 ± 0.0000	0.2845 ± 0.0000
Puma32	0.9897 ± 0.0000	0.9566 ± 0.0000	0.9599 ± 0.0000	0.9885 ± 0.0000	0.9904 ± 0.0000	0.9675 ± 0.0000	0.9672 ± 0.0000
Telemonitoring	0.3676 ± 0.0000	0.9573 ± 0.0000	0.9876 ± 0.0000	1.0209 ± 0.0000	0.9942 ± 0.0000	0.9924 ± 0.0000	0.9547 ± 0.0000
Volatility	0.3736 ± 0.0000	1.4429 ± 0.0000	0.8578 ± 0.0000	0.7670 ± 0.0000	0.7670 ± 0.0000	0.7551 ± 0.0000	0.7135 ± 0.0000
Average rank		3.40 ± 1.80	2.80 ± 1.25	4.50 ± 1.20	3.50 ± 1.50	5.00 ± 1.26	1.80 ± 0.87

Table 17: RMSE scores of random forest models trained with DP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.4141 ± 0.0000	0.6298 ± 0.0000	0.5883 ± 0.0000	0.5531 ± 0.0000	0.4826 ± 0.0000	0.5472 ± 0.0000	0.4701 ± 0.0000
Compactiv	0.1514 ± 0.0000	0.1376 ± 0.0000	0.1325 ± 0.0000	0.1250 ± 0.0000	0.1223 ± 0.0000	0.3158 ± 0.0000	0.1257 ± 0.0000
Cpusmall	0.1564 ± 0.0000	0.1485 ± 0.0000	0.1467 ± 0.0000	0.1442 ± 0.0000	0.1446 ± 0.0000	0.3391 ± 0.0000	0.1438 ± 0.0000
Ctscan	0.0345 ± 0.0000	0.1971 ± 0.0000	0.1616 ± 0.0000	0.0852 ± 0.0000	0.0838 ± 0.0000	0.1165 ± 0.0000	0.0847 ± 0.0000
Indoorloc	0.0416 ± 0.0000	0.2384 ± 0.0000	0.0889 ± 0.0000	0.0586 ± 0.0000	0.0588 ± 0.0000	0.0899 ± 0.0000	0.0577 ± 0.0000
Mv	0.0443 ± 0.0000	0.0371 ± 0.0000	0.0366 ± 0.0000	0.0405 ± 0.0000	0.0402 ± 0.0000	0.0860 ± 0.0000	0.0365 ± 0.0000
Pole	0.1357 ± 0.0000	0.1287 ± 0.0000	0.1235 ± 0.0000	0.1105 ± 0.0000	0.1114 ± 0.0000	1.9624 ± 0.0000	0.1100 ± 0.0000
Puma32	0.7866 ± 0.0000	0.7576 ± 0.0000	0.7611 ± 0.0000	0.7860 ± 0.0000	0.7852 ± 0.0000	0.7656 ± 0.0000	0.7646 ± 0.0000
Telemonitoring	0.1971 ± 0.0000	0.7872 ± 0.0000	0.7896 ± 0.0000	0.6094 ± 0.0000	0.5861 ± 0.0000	0.5925 ± 0.0000	0.5724 ± 0.0000
Volatility	0.2173 ± 0.0000	1.3070 ± 0.0000	0.5174 ± 0.0000	0.3624 ± 0.0000	0.3518 ± 0.0000	0.3542 ± 0.0000	0.3363 ± 0.0000
Average rank		4.80 ± 1.54	4.10 ± 1.22	3.40 ± 1.36	2.60 ± 1.20	4.60 ± 1.28	1.50 ± 0.81

Table 18: MAE scores of random forest models trained with DP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.5669 ± 0.0000	0.8079 ± 0.0000	0.8052 ± 0.0000	0.7877 ± 0.0000	0.7476 ± 0.0000	0.7711 ± 0.0000	0.7117 ± 0.0000
Compactiv	0.1741 ± 0.0000	0.4182 ± 0.0000	0.3555 ± 0.0000	0.2544 ± 0.0000	0.3564 ± 0.0000	0.2524 ± 0.0000	0.3117 ± 0.0000
Cpusmall	0.1803 ± 0.0000	0.5635 ± 0.0000	0.7114 ± 0.0000	0.3377 ± 0.0000	0.3396 ± 0.0000	0.3364 ± 0.0000	0.3211 ± 0.0000
Ctscan	0.1317 ± 0.0000	0.7413 ± 0.0000	0.8081 ± 0.0000	0.6240 ± 0.0000	0.3193 ± 0.0000	0.6205 ± 0.0000	0.3031 ± 0.0000
Indoorloc	0.0657 ± 0.0000	0.9650 ± 0.0000	0.9799 ± 0.0000	0.3990 ± 0.0000	0.3131 ± 0.0000	0.3987 ± 0.0000	0.2866 ± 0.0000
Mv	0.0478 ± 0.0000	0.5557 ± 0.0000	0.5491 ± 0.0000	1.2969 ± 0.0000	0.7808 ± 0.0000	1.4522 ± 0.0000	0.6575 ± 0.0000
Pole	0.2146 ± 0.0000	0.5366 ± 0.0000	0.4636 ± 0.0000	0.3137 ± 0.0000	0.3185 ± 0.0000	0.6767 ± 0.0000	0.3025 ± 0.0000
Puma32	1.0047 ± 0.0000	0.9881 ± 0.0000	0.9873 ± 0.0000	1.0128 ± 0.0000	1.0130 ± 0.0000	0.9939 ± 0.0000	0.9939 ± 0.0000
Telemonitoring	0.4610 ± 0.0000	0.8693 ± 0.0000	0.8708 ± 0.0000	1.2163 ± 0.0000	1.1387 ± 0.0000	1.1816 ± 0.0000	1.0264 ± 0.0000
Volatility	0.5938 ± 0.0000	0.6589 ± 0.0000	0.6980 ± 0.0000	0.7912 ± 0.0000	0.7844 ± 0.0000	0.7732 ± 0.0000	0.7476 ± 0.0000
Average rank		3.80 ± 1.94	3.70 ± 1.95	4.10 ± 1.37	3.70 ± 1.35	3.70 ± 1.55	2.00 ± 1.00

Table 19: RMSE scores of random forest models trained with BP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.3853 ± 0.0000	0.6634 ± 0.0000	0.6419 ± 0.0000	0.4719 ± 0.0000	0.4728 ± 0.0000	0.4635 ± 0.0000	0.4615 ± 0.0000
Compactiv	0.1089 ± 0.0000	0.3374 ± 0.0000	0.2907 ± 0.0000	0.1340 ± 0.0000	0.2224 ± 0.0000	0.1329 ± 0.0000	0.2070 ± 0.0000
Cpusmall	0.1105 ± 0.0000	0.2150 ± 0.0000	0.2389 ± 0.0000	0.1393 ± 0.0000	0.1402 ± 0.0000	0.1384 ± 0.0000	0.1369 ± 0.0000
Ctscan	0.0430 ± 0.0000	0.5514 ± 0.0000	0.5712 ± 0.0000	0.2349 ± 0.0000	0.0890 ± 0.0000	0.2343 ± 0.0000	0.1009 ± 0.0000
Indoorloc	0.0392 ± 0.0000	0.8875 ± 0.0000	0.6489 ± 0.0000	0.2112 ± 0.0000	0.1381 ± 0.0000	0.2117 ± 0.0000	0.1459 ± 0.0000
Mv	0.0309 ± 0.0000	0.4178 ± 0.0000	0.2488 ± 0.0000	0.8571 ± 0.0000	0.3576 ± 0.0000	1.0755 ± 0.0000	0.3308 ± 0.0000
Pole	0.0804 ± 0.0000	0.4787 ± 0.0000	0.3425 ± 0.0000	0.1089 ± 0.0000	0.1120 ± 0.0000	0.3939 ± 0.0000	0.1160 ± 0.0000
Puma32	0.7992 ± 0.0000	0.7772 ± 0.0000	0.7784 ± 0.0000	0.8005 ± 0.0000	0.8031 ± 0.0000	0.7834 ± 0.0000	0.7838 ± 0.0000
Telemonitoring	0.2700 ± 0.0000	0.7189 ± 0.0000	0.6828 ± 0.0000	0.7659 ± 0.0000	0.6941 ± 0.0000	0.7473 ± 0.0000	0.6371 ± 0.0000
Volatility	0.3691 ± 0.0000	0.3914 ± 0.0000	0.3968 ± 0.0000	0.4350 ± 0.0000	0.4345 ± 0.0000	0.4275 ± 0.0000	0.4161 ± 0.0000
Average rank		4.40 ± 1.85	3.80 ± 1.78	3.80 ± 1.60	3.30 ± 1.55	3.50 ± 1.50	2.20 ± 0.98

Table 20: MAE scores of random forest models trained with BP data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.5683 ± 0.0000	0.8131 ± 0.0000	0.8147 ± 0.0000	0.7944 ± 0.0000	0.7586 ± 0.0000	0.7809 ± 0.0000	**0.7222 ± 0.0000**
Compactiv	0.1826 ± 0.0000	0.4521 ± 0.0000	0.3587 ± 0.0000	0.2540 ± 0.0000	0.3644 ± 0.0000	0.2533 ± 0.0000	0.3226 ± 0.0000
Cpusmall	0.1855 ± 0.0000	0.5622 ± 0.0000	0.7065 ± 0.0000	0.3648 ± 0.0000	0.3658 ± 0.0000	0.3635 ± 0.0000	**0.3500 ± 0.0000**
Ctscan	0.1299 ± 0.0000	0.7458 ± 0.0000	0.8175 ± 0.0000	0.6050 ± 0.0000	0.3275 ± 0.0000	0.6018 ± 0.0000	**0.2982 ± 0.0000**
Indoorloc	0.0660 ± 0.0000	0.9636 ± 0.0000	0.9679 ± 0.0000	0.3693 ± 0.0000	0.2918 ± 0.0000	0.3665 ± 0.0000	**0.2603 ± 0.0000**
Mv	0.0481 ± 0.0000	0.5548 ± 0.0000	0.5484 ± 0.0000	1.2959 ± 0.0000	0.7611 ± 0.0000	1.4976 ± 0.0000	0.6415 ± 0.0000
Pole	0.2109 ± 0.0000	0.5313 ± 0.0000	0.4618 ± 0.0000	0.3114 ± 0.0000	0.3252 ± 0.0000	0.6804 ± 0.0000	**0.3039 ± 0.0000**
Puma32	1.0054 ± 0.0000	0.9890 ± 0.0000	0.9901 ± 0.0000	1.0131 ± 0.0000	1.0187 ± 0.0000	0.9927 ± 0.0000	0.9947 ± 0.0000
Telemonitoring	0.4511 ± 0.0000	0.8723 ± 0.0000	0.8704 ± 0.0000	1.2799 ± 0.0000	1.2418 ± 0.0000	1.2461 ± 0.0000	1.1443 ± 0.0000
Volatility	0.4741 ± 0.0000	0.7608 ± 0.0000	0.6469 ± 0.0000	0.7255 ± 0.0000	0.7154 ± 0.0000	0.7014 ± 0.0000	0.6780 ± 0.0000
Average rank		4.20 ± 1.72	3.70 ± 2.15	4.00 ± 1.26	3.60 ± 1.28	3.50 ± 1.57	2.00 ± 1.10

Table 21: RMSE scores of random forest models trained with KMeans data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.3871 ± 0.0000	0.6676 ± 0.0000	0.6510 ± 0.0000	0.4748 ± 0.0000	0.4810 ± 0.0000	0.4673 ± 0.0000	0.4706 ± 0.0000
Compactiv	0.1106 ± 0.0000	0.3589 ± 0.0000	0.2937 ± 0.0000	0.1349 ± 0.0000	0.2312 ± 0.0000	0.1345 ± 0.0000	0.2165 ± 0.0000
Cpusmall	0.1112 ± 0.0000	0.2148 ± 0.0000	0.2385 ± 0.0000	0.1423 ± 0.0000	0.1445 ± 0.0000	0.1414 ± 0.0000	**0.1401 ± 0.0000**
Ctscan	0.0424 ± 0.0000	0.5540 ± 0.0000	0.5784 ± 0.0000	0.2288 ± 0.0000	0.0928 ± 0.0000	0.2283 ± 0.0000	0.1008 ± 0.0000
Indoorloc	0.0393 ± 0.0000	0.8864 ± 0.0000	0.6439 ± 0.0000	0.1916 ± 0.0000	0.1179 ± 0.0000	0.1908 ± 0.0000	0.1319 ± 0.0000
Mv	0.0310 ± 0.0000	0.4157 ± 0.0000	0.2487 ± 0.0000	0.8557 ± 0.0000	0.3417 ± 0.0000	1.1142 ± 0.0000	0.3168 ± 0.0000
Pole	0.0804 ± 0.0000	0.4756 ± 0.0000	0.3440 ± 0.0000	0.1099 ± 0.0000	0.1164 ± 0.0000	0.4005 ± 0.0000	0.1186 ± 0.0000
Puma32	0.7966 ± 0.0000	0.7766 ± 0.0000	0.7785 ± 0.0000	0.7994 ± 0.0000	0.8036 ± 0.0000	0.7821 ± 0.0000	0.7841 ± 0.0000
Telemonitoring	0.2677 ± 0.0000	0.7229 ± 0.0000	0.6829 ± 0.0000	0.8430 ± 0.0000	0.7840 ± 0.0000	0.8266 ± 0.0000	0.7344 ± 0.0000
Volatility	0.2360 ± 0.0000	0.4709 ± 0.0000	0.3877 ± 0.0000	0.3338 ± 0.0000	0.4375 ± 0.0000	0.3242 ± 0.0000	0.4081 ± 0.0000
Average rank		4.70 ± 1.73	3.80 ± 1.83	3.50 ± 1.50	3.40 ± 1.56	3.00 ± 1.73	2.60 ± 0.92

Table 22: MAE scores of random forest models trained with KMeans data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.5724 ± 0.0026	0.8726 ± 0.0203	0.8566 ± 0.0231	0.7755 ± 0.0513	0.7584 ± 0.0527	0.7502 ± 0.0501	0.7231 ± 0.0536
Compactiv	0.2066 ± 0.0106	0.4396 ± 0.0920	0.3940 ± 0.0390	0.2735 ± 0.0108	0.3287 ± 0.0371	0.2715 ± 0.0108	0.2996 ± 0.0290
Cpusmall	0.1842 ± 0.0022	0.5506 ± 0.0163	0.6610 ± 0.0313	0.3404 ± 0.0059	0.3429 ± 0.0073	0.3402 ± 0.0059	0.3324 ± 0.0102
Ctscan	0.1358 ± 0.0021	0.7694 ± 0.0119	0.8689 ± 0.0240	0.6316 ± 0.0403	0.5015 ± 0.2038	0.6276 ± 0.0400	0.4485 ± 0.2154
Indoorloc	0.0658 ± 0.0004	0.9605 ± 0.0070	0.9735 ± 0.0117	0.5531 ± 0.1698	0.5120 ± 0.1814	0.5469 ± 0.1714	0.4534 ± 0.1663
Mv	0.0884 ± 0.0067	0.6094 ± 0.0196	0.5360 ± 0.0182	1.2948 ± 0.0009	0.4814 ± 0.0910	1.3253 ± 0.0081	0.4320 ± 0.0917
Pole	0.2144 ± 0.0008	0.5428 ± 0.0029	0.4567 ± 0.0081	0.3209 ± 0.0115	0.3197 ± 0.0086	0.6693 ± 0.0059	0.3030 ± 0.0033
Puma32	1.0023 ± 0.0028	0.9855 ± 0.0011	0.9873 ± 0.0011	1.0179 ± 0.0027	1.0166 ± 0.0044	0.9941 ± 0.0016	0.9935 ± 0.0017
Telemonitoring	0.4528 ± 0.0087	0.8792 ± 0.0052	0.8663 ± 0.0136	0.8894 ± 0.0926	0.8593 ± 0.1118	0.8498 ± 0.0884	0.7749 ± 0.1097
Volatility	0.4880 ± 0.0000	0.6351 ± 0.0000	0.6410 ± 0.0000	0.6764 ± 0.0000	0.6777 ± 0.0000	0.6570 ± 0.0000	0.6319 ± 0.0000
Average rank		4.40 ± 1.56	4.40 ± 1.36	4.20 ± 1.25	3.30 ± 1.35	3.30 ± 1.62	1.40 ± 0.80

Table 23: RMSE scores of random forest models trained with A-KMeans data

Dataset	Oracle	Mean	Median	D-SEL(u)	D-SEL(u')	D-ENS(u)	Proposed method
Bikesharing	0.3941 ± 0.0021	0.7392 ± 0.0276	0.7089 ± 0.0284	0.4766 ± 0.0271	0.4816 ± 0.0273	0.4640 ± 0.0259	0.4730 ± 0.0274
Compactiv	0.1149 ± 0.0017	0.3642 ± 0.0739	0.3217 ± 0.0407	0.1377 ± 0.0029	0.1838 ± 0.0214	0.1371 ± 0.0029	0.1774 ± 0.0204
Cpusmall	0.1137 ± 0.0014	0.2444 ± 0.0223	0.2432 ± 0.0069	0.1415 ± 0.0010	0.1403 ± 0.0024	0.1406 ± 0.0010	0.1375 ± 0.0016
Ctscan	0.0463 ± 0.0007	0.5679 ± 0.0083	0.6257 ± 0.0188	0.2577 ± 0.0300	0.1964 ± 0.1473	0.2567 ± 0.0297	0.1901 ± 0.1492
Indoorloc	0.0395 ± 0.0001	0.8858 ± 0.0089	0.6650 ± 0.0352	0.2374 ± 0.0890	0.1960 ± 0.0826	0.2370 ± 0.0892	0.2030 ± 0.0739
Mv	0.0431 ± 0.0019	0.4987 ± 0.0223	0.3388 ± 0.0199	0.8534 ± 0.0012	0.1922 ± 0.0471	0.9160 ± 0.0122	0.1858 ± 0.0471
Pole	0.0812 ± 0.0006	0.4827 ± 0.0020	0.3367 ± 0.0099	0.1113 ± 0.0030	0.1122 ± 0.0026	0.3805 ± 0.0061	0.1175 ± 0.0018
Puma32	0.7968 ± 0.0026	0.7756 ± 0.0010	0.7782 ± 0.0009	0.8043 ± 0.0037	0.8044 ± 0.0044	0.7828 ± 0.0014	0.7827 ± 0.0014
Telemonitoring	0.2795 ± 0.0041	0.7319 ± 0.0066	0.6842 ± 0.0115	0.5139 ± 0.0614	0.4940 ± 0.0724	0.4995 ± 0.0579	0.4727 ± 0.0669
Volatility	0.2523 ± 0.0000	0.3878 ± 0.0000	0.3680 ± 0.0000	0.3223 ± 0.0000	0.3604 ± 0.0000	0.3158 ± 0.0000	0.3357 ± 0.0000
Average rank		5.20 ± 1.54	4.50 ± 1.12	3.40 ± 1.28	2.90 ± 1.45	3.00 ± 1.61	2.00 ± 0.89

Table 24: MAE scores of random forest models trained with A-KMeans data