Post-COVID Stock Forecasting: An Investigation of Bayesian Methods in Machine Learning

Renyi Qu
Faculty of Economics, Keio University
July 5th, 2021

Abstract

The COVID-19 pandemic has caused severe damage to the global economy and raised extreme uncertainties in the stock markets. This research intends to investigate Bayesian methods in machine learning for stock index forecasting in the post-COVID era. First, exploratory analysis for the selected time series data will be conducted. Second, the general idea of Bayesian methods will be introduced, and the selected models will be discussed in theoretical details. Third, the experimental results will be discussed to draw general conclusions for further research.

1. Introduction

The COVID-19 pandemic has caused severe damage to the global economy and raised extreme uncertainties in the stock markets. In fact, Baker et al. (2020) pointed out that the COVID-19 shock is "unprecedented" throughout human history. The most heavily affected victim is the global airline industry. Based on the analysis in Baker et al.'s paper, the year-to-year change in flight frequency dropped by nearly 75% from January to May in 2020, resulting in a huge loss in the revenue of airline

companies. Nevertheless, some industries actually obtained enormous benefits from the pandemic, such as the pharmaceutical industry and the advanced technologies industry (Nicola et al., 2020). After the pandemic spread around the globe, the global financial markets became extremely volatile, and no one had a clear clue of what was going to happen next (Baker et al., 2020). Hence, stock trading became a highly challenging task. Owing to the exponential growth of data availability in the past decade, data-driven approaches (i.e. machine learning) became more and more appealing towards investors and researchers, but the huge dependency on data has also become a major bottleneck of these approaches (Lemke et al., 2015). To tackle such issues and innovate forward, researchers have attempted to integrate Bayesian methods into these approaches for time series forecasting and stock trading.

This research intends to investigate the functionality and performance of Bayesian methods in machine learning for stock index forecasting in the post-COVID era. Chapter 2 focuses on the exploratory analysis for the selected stock indices. Chapter 3 illustrates the theoretical backgrounds in two streams. First, the general idea of Bayesian methods will be introduced. Second, the selected machine learning models will be discussed in theoretical details, together with their corresponding Bayesian integration approaches. Chapter 4 discusses the experimental settings on post-COVID stock index forecasting, and Chapter 5 examines the experimental results. In the end, Chapter 6 concludes the research.

2. Data

The dataset is obtained from Yahoo Finance, including the daily adjusted close prices of 4 selected stock indices in 4 different regions as shown in Table 1. The goal is to build Bayesian machine learning algorithms to capture the trend from the historical daily prices (2011/06/01 - 2019/06/01) in order to forecast the prices during the globalization of the pandemic (2019/06/01 - 2021/06/01).

Table 1: Dataset description

Asset class	Symbol	Full name	Price Range
Equity Index	^N225	Nikkei 225	[8160.01, 30467.75]
Equity Index	^IXIC	NASDAQ Composite	[2335.83, 14138.78]
Equity Index	000001.SS	SSE Composite Index	[1950.01, 5166.35]
Equity Index	^HSI	HANG SENG INDEX	[16250.27, 33154.12]

The following figure visualizes the normalized prices of these indices during the COVID-19 pandemic:

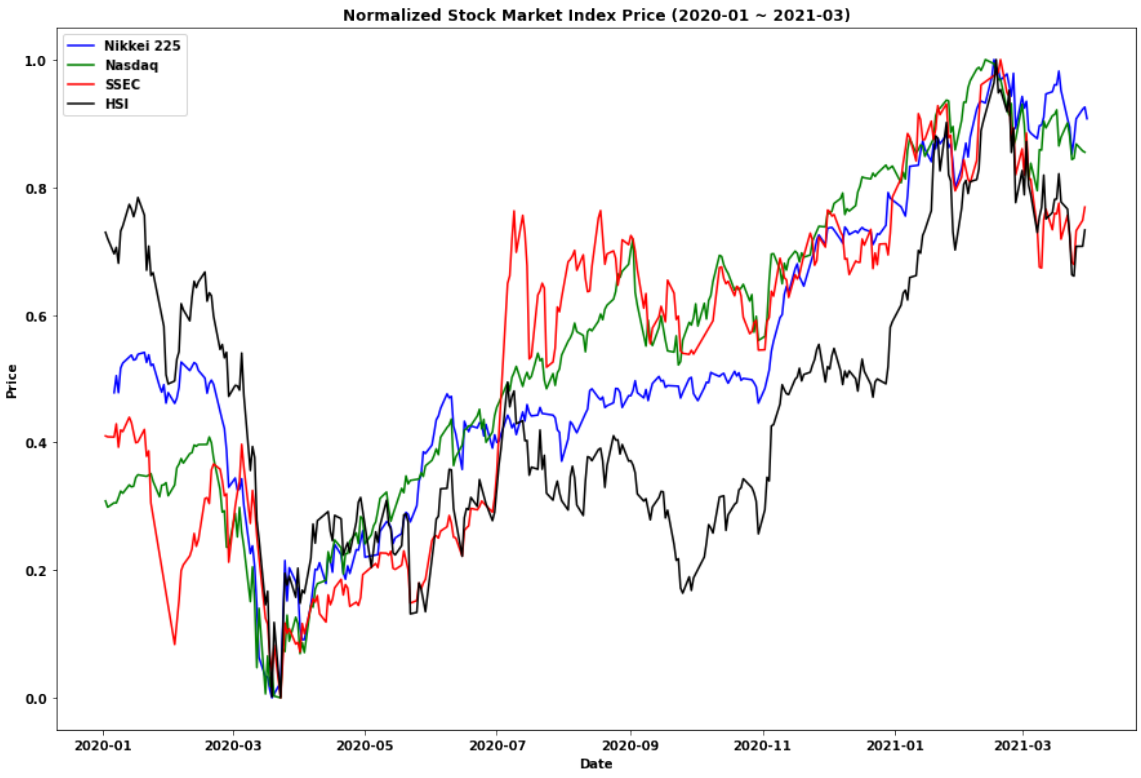

Figure 1. Normalized Stock Market Index Price (2020/01-2021/03)

The pandemic first broke out in China in January, leading to an immediate shock in the Chinese stock market as shown in red. As the pandemic spread all across the

globe during March, a dramatic decline took place in all the 4 stock indices. The Hang Seng Index, representing one of the most globalized stock markets, suffered the highest amount of decline in normalized price according to Figure 1. As the minimum was reached in late March, global markets gradually became accustomed to the pandemic, and the selected stock indices gradually recovered throughout 2020 but seemed to decline again in 2021.

The Augmented Dickey-Fuller test (ADF) was used to test the stationarity of time series (i.e. whether the statistical properties of the time series stay the same across time) (Hamilton, 2020). Since stock prices are notorious for non-stationarity (Chen et al., 2008), the p-values for ADF statistics were extremely high and the null hypothesis of non-stationarity was not rejected. After applying first-order differencing, all the 4 series became stationary, but the autocorrelation (ACF) and partial autocorrelation function (PACF) indicated no visible pattern of autoregression or moving average. Similarly, rescaling the price data into returns data did not show promising ACF and PACF results either. After applying second-order differencing, we finally obtained crucial clues for autocorrelation patterns, as shown in Figure 2. Since the 2nd-order difference is a discrete analogy to the 2nd-order derivative for a continuous function, this result shows that the curvature of stock index time series during the pandemic is autocorrelated (Hamilton, 2020).

Table 2: ADF Test Results

Index	Original	1st-order difference	2nd-order difference
Nikkei225	-0.483	-10.457	-8.759
	(0.895)	(0.000)	(0.000)
NASDAQ	-0.746	-5.049	-12.555
	(0.834)	(0.000)	(0.000)
SSEC	-1.197	-16.587	-10.204
	(0.675)	(0.000)	(0.000)
HSI	-1.525	-11.783	-8.499
	(0.521)	(0.000)	(0.000)

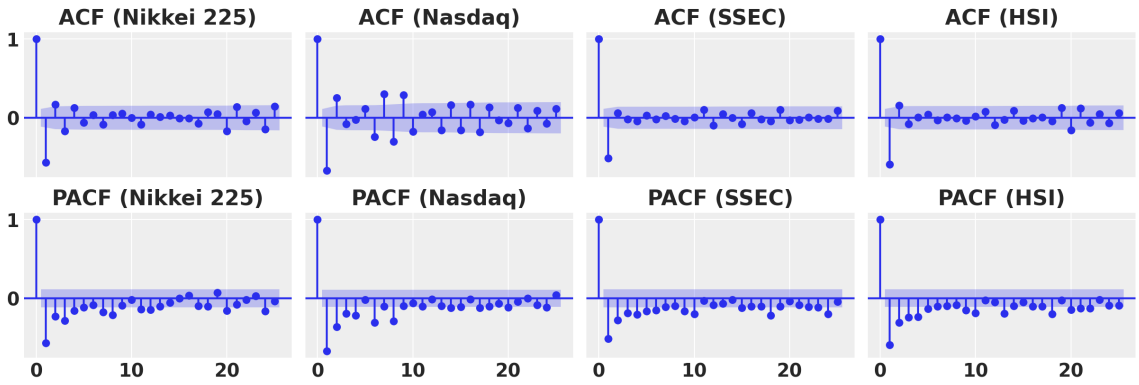

Figure 2. 2nd-order difference autocorrelation results

3. Methods

In Chapter 3.1, we introduce the basics of Bayesian methods, the two posterior inference branches (Markov Chain Monte Carlo and Variational Inference), and Bayesian optimization for hyperparameter tuning. In this project, Bayesian optimization for hyperparameter tuning is only applied to decision tree ensembles models. For parameter inference, Variational Inference is used in neural networks, and MCMC is used in Support Vector Regression and Bayesian Predictive Synthesis. In Chapter 3.2, we discuss the basics of each model separately and examine how Bayesian methods can be applied to each one.

3.1 Intuition of Bayesian Methods

All Bayesian methods center around the Bayes' Theorem (Bayes, 1763):

$$p(\theta|x) = \frac{p(\theta)p(x|\theta)}{p(x)}$$

where

- **Prior**: $p(\theta)$ is a subjective assumption of θ, determined by its probability distribution and the corresponding hyperparameters.

- **Likelihood**: $p(x|\theta)$ is a measure of extent to which the assumption of θ provides support for the particular distribution of information x. This is the underlying data-generating model.
- **Denominator**: $p(x)$ is a function of data x and is not associated with our target θ, so it works as a normalization constant. However, this constant is typically not given to us and has to be calculated by marginalizing the joint distribution between the parameter and the data: $p(x) = \int p(x|\theta)p(\theta)d\theta$.
- **Posterior**: $p(\theta|x)$ is the updated belief of θ by the given information x.

Statisticians are only concerned with two parts when using Bayesian methods: modeling and computing (Betancourt, 2017). In modeling, we define prior and formulate likelihood. In computing, we generate posterior based on prior and likelihood. The computing process is the most crucial part, because it is extremely difficult, if not impossible, to directly compute posteriors even with computers, especially when the number of dimensions (i.e. number of parameters) gets higher. There are only a limited amount of conjugate distributions (i.e. prior and posterior follow the same distribution) that allows statisticians to derive the closed-form posterior formulas with ease. For non-conjugate priors, posterior integration is painful.

Markov Chain Monte Carlo

Fortunately, Gelfand and Smith (1990) managed to tackle this issue by introducing Markov Chain Monte Carlo (MCMC), the most common posterior inference method in Bayesian analysis. MCMC is a combination of Markov Chain sampling and Monte Carlo integration. Markov Chain sampling generates a sequence of random samples where the current state is only dependent on the previous state, while Monte Carlo integration uses numerical approximation to generate various properties from the samples as posterior statistics. There is a handful number of MCMC samplers available as summarized in van de Schoot et al. (2021)'s paper, and the most basic one is the Metropolis–Hastings (MH) algorithm (Chib and Greenberg, 1995):

Algorithm 1: Metropolis-Hastings algorithm

Initialize: $t = 0$, x_0, $g(x|y)$;
for t *in* $[1, t_{max}]$ **do**
  Generate a random candidate $x \sim g(x|x_t)$;
  Calculate acceptance probability $a = \min\left(1, \frac{P(x)}{P(x_t)} \frac{g(x_t|x)}{g(x|x_t)}\right)$;
  Generate a Bernoulli indicator $u \sim \text{Bernoulli}(a)$;
  if u **then**
    $x_{t+1} = x$;
  else
    $x_{t+1} = x_t$;
  end
end

where

- $g(x|y)$: an arbitrary proposal density function (i.e. prior), typically set to $N(y, \sigma^2)$.
- $P(x)$: desired density distribution (i.e. likelihood).
- $P(y) \times g(x|y)$: posterior distribution of value y.
- $r(x, x_t) = \frac{P(x)}{P(x_t)} \frac{g(x_t|x)}{g(x|x_t)}$: posterior odds ratio between candidate x and previous value x_t. If this ratio is higher than 1, then candidate x is more likely than x_t at the next step, and we accept this candidate. If this ratio is lower than 1, rather than immediately reject the candidate, we impose a uniform randomness into the decision - at a probability of a we accept and otherwise we reject.

Based on the MH algorithm, Hamiltonian Monte Carlo (HMC) algorithm was created in order to minimize the correlation between samples. It uses the Hamiltonian of the system in place of the original posterior distribution (Jospin et al., 2020):

$$H(x, v) = \ln P(x) + \ln Q(v)$$

where $\ln P(x)$ is considered the potential energy of the system (i.e. underlying likelihood model) and $\ln Q(v)$ is considered the kinetic energy of the system (i.e.

random noise introduced by v). With that, a new acceptance probability is established:

$$a = \min\left(1, \frac{\exp\left(H(x_T, v_T)\right)}{\exp\left(H(x_0, v_0)\right)}\right)$$

where T is the integration time. To automatically optimize the T in HMC algorithm, Hoffman and Gelman (2014) further introduced No-U-Turn Sampler, which is now widely used in many probabilistic programming packages. In this project, the MH-algorithm will be used in Support Vector Regression, while a more sophisticated Gibbs sampler will be used in Bayesian Predictive Synthesis following their original code (McAlinn and West, 2019).

Variational Inference

Another unique branch for posterior inference is Variational Inference (VI), which is vastly used in Bayesian deep learning in place of MCMC due to its flexible scalability (Jospin et al., 2020). The idea behind VI is to approximate the posterior distribution $p(H|D)$ with $q(H)$ and to make them as similar as possible. H is the hypothesis we are interested in, and D is the dataset. This similarity is typically measured by Kullback–Leibler (KL) divergence (Blei et al., 2017):

$$D_{\mathrm{KL}}(q\|p) = \int_H q(H') \ln \frac{q(H')}{p(H'|D)} dH'$$

Note that the posterior of our interest $p(H'|D)$ is still in the equation, so it seems pointless to compute KL-divergence if we need to compute the posterior first. Nevertheless, if we derive further, we obtain a very essential equation for VI method:

$$\begin{aligned}
D_{\mathrm{KL}}(q\|p) &= \int_H q(H')\Big(\ln \frac{q(H')}{p(H', D)} + \ln p(D)\Big) dH' \\
&= \int_H q(H') \ln \frac{q(H')}{p(H', D)} dH' + \int_H q(H') \ln p(D) dH' \\
&= \mathbb{E}_q[\ln q(H) - \ln q(H, D)] + \ln p(D) \\
&= \ln p(D) - \mathcal{L}(q)
\end{aligned}$$

Since $\ln p(D)$ is a fixed constant, to minimize $D_{\mathrm{KL}}(q\|p)$ is to maximize $\mathcal{L}(q)$. Therefore, we simply reduced the objective of VI to a maximization problem, and this $\mathcal{L}(q)$ is

called Evidence Lower Bound (ELBO) (Blei et al., 2017). The applicability of VI in Bayesian Neural Network will be discussed in the corresponding subsection.

Bayesian Optimization for Hyperparameter Tuning

In addition to parameter inference, Bayesian methods are widely used to tune the hyperparameters for various machine learning algorithms (Snoek et al., 2012). The basic procedure goes as follows:

1. Define a hyperparameter space $\mathcal{H}$ for all hyperparameters of interest.
2. Initialize a sample set $\{\mathbf{h}_j, l_j\}_{j=1}^{N}$, where N is the total amount of initial samples, $\mathbf{h}_j \in \mathcal{H}$ are the hyperparameter samples, and l_j is the target function that we would like to optimize, associated with the accuracy of the model (e.g. a loss function).
3. Train a Gaussian Process Regressor (discussed in Chapter 3.2) based on the sample set with a Gaussian prior $f(\mathbf{h})$ and thus the output $l_j \sim N(f(\mathbf{h}_j), v)$, where v is a random noise.
4. Identify the hyperparameter that minimizes the acquisition function. There are researches done on the design of various acquisition functions, but Snoek et al. (2012) identified GP lower confidence bound as the optimal one so far:

$$a_{\text{LCB}}(\mathbf{h}; \{\mathbf{h}_j, l_j\}) = \mu(\mathbf{h}; \{\mathbf{h}_j, l_j\}) - \kappa\sigma(\mathbf{h}; \{\mathbf{h}_j, l_j\})$$

   where μ and σ are the mean and standard deviation of $f(\mathbf{h})$ respectively, and κ can be tuned to balance exploration and exploitation.
5. Add $\mathbf{h}_{\text{opt}} = \arg\min_{\mathbf{h}} a_{\text{LCB}}(\mathbf{h}; \{\mathbf{h}_j, l_j\})$ back to the sample set.
6. Repeat 3-5 until convergence.

Due to the time constraint of this project, we will only apply Bayesian optimization to the hyperparameter tuning of decision tree ensembles models.

3.2 Models

This section introduces the functionality of the following models separately:

- Regression: Gaussian Process Regression, Support Vector Regression
- Neural networks: Bayesian Neural Network, Long Short-Term Memory
- Decision tree ensembles: Random Forest, XGBoost
- Pure Bayesian: Bayesian Predictive Synthesis

To apply regression models, neural networks, and boosting methods on time series data, we rearrange the data in a supervised learning manner so that at each time t, the current value x_t is the dependent variable while the lagged values $x_{t-1}, \cdots, x_{t-p}$ are the explanatory variables. In other words, we explicitly focus on the autocorrelation within the time series itself without involving any exogenous variable.

Gaussian Process Regression

Gaussian Process Regression (GPR) is a fundamental, non-parametric Bayesian method. Unlike most regression models, GPR is not a single function to indicate the relationship between the dependent variable and the explanatory variables. Instead, a Gaussian Process is a collection of random functions that are normally distributed. We impose the following latent multivariate normal distribution (Roberts et al., 2013):

$$f(x) \sim \mathcal{GP}(\mu(x), k(x, x'))$$

where

- $f(x)$: latent function of x following multivariate normal distribution (i.e. Gaussian Process).
- $\mu(x)$: mean function.
- $k(x, x')$: covariance function (i.e. kernel).

The output will be the latent function with a Gaussian noise $\varepsilon_y \sim N(0, \sigma_y^2)$:

$$y \sim \mathcal{GP}(\mu(x), k(x, x') + \sigma_y^2 I)$$

For predictive posterior inference, we set up the following joint normal distribution:

$$\begin{bmatrix} f(x) \\ f(x^*) \end{bmatrix} \sim N\left(\begin{bmatrix} \mu(x) \\ \mu(x^*) \end{bmatrix}, \begin{bmatrix} k(x,x') & k(x^*,x) \\ k(x^*,x) & k(x^*,x^{*'}) \end{bmatrix} \right)$$

where x^* is the new data point. Then we can obtain a conditional posterior from:

$$f(x^*)|f(x) \sim N(\mu^*, \sigma^{*^2})$$

where

- $\mu^* = \mu(x^*) + k(x^*,x)k(x,x')^{-1}(y - \mu(x))$
- $\sigma^{*2} = k(x^*,x^{*'}) - k(x^*,x)k(x,x')^{-1}k(x,x^*)$

There have been many researches done on different kernel choices (Roberts et al., 2013), and we choose the most commonly used radial basis function (RBF) assuming constant variance:

$$k(x,x') = \sigma_f^2 e^{-\frac{\|x-x'\|^2}{2l^2}}$$

where

- σ_f^2: constant variance of $f(x)$ as vertical amplitude.
- l: length-scale that measures the (horizontal) extent of decrease in the correlation with respect to the increase in the distance between the two data points.
- $\|x - x'\|$: Euclidean distance between given input x and a fixed point x'.
- $\frac{1}{2l^2}$: shape parameter to normalize the input for RBF.

Support Vector Regression

Support Vector Machine (SVM) is a classification method that tries to construct a decision boundary with maximal geometric margin in order to obtain accurate classification results. In other words, SVM tries to solve an optimization problem:

$$\arg\min_f \gamma R(f) + \sum_i \mathcal{L}(y_i f(x_i))$$

where

- $f(x)$: latent function of data x.

- $R(f)$: regularization function to penalize overfitting.
- γ: hyperparameter to adjust the effect of regularization function.
- $\mathcal{L}(m_i)$: loss function based on geometric margin m_i.
- y_i: actual output matching input x_i.

Drucker et al. (1997) modified SVM into Support Vector Regression (SVR) so that it can be applied to regression problems, and empirical evidence shows the relatively good performance of it compared to other regression methods. The ϵ-SVR is:

$$\arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i |\xi_i| + \sum_i \mathcal{L}(y_i - f(\mathbf{w}, x_i))$$

where

- C: tolerance hyperparameter. The higher C, the more data points outside the margin of decision boundary are included.
- ξ_i: deviation of the ith data point from the margin of decision boundary.
- $R(f) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i |\xi_i|$: a combination of L2 penalty and deviation tolerance.
- $\mathcal{L} = \max(0, |y_i - f(\mathbf{w}, x_i)| - (\epsilon + |\xi_i|))$: this loss function aims to constraint the errors below the predefined ϵ plus the deviation $|\xi_i|$.

The Bayesian integration of SVM/SVR with constant hyperparameter is (Law and Kwok, 2001):

$$p(f|D) \propto p(f)p(D|f)$$

where D is the dataset and

$$p(f) = N(f; 0, k(x, x')) = \frac{1}{\sqrt{2\pi \det k(x, x')}} \exp\Big\{ -\frac{1}{2} f^T k^{-1}(x, x') f \Big\}$$

$$p(D|f) = \prod_i p(y_i - f(x_i)) = \frac{C}{2(C\epsilon + 1)} \exp\Big\{ -C\sum_i \mathcal{L}(y_i - f(\mathbf{w}, x_i)) \Big\}$$

and thus

$$p(f|D) \propto \exp\Big\{ -C\sum_i \mathcal{L}(y_i - f(\mathbf{w}, x_i)) - \frac{1}{2} f^T k^{-1}(x, x') f \Big\}$$

Since we do not have a closed-form formula for the posterior, we use MCMC for posterior sampling. Specifically, we use the MH-algorithm.

Bayesian Neural Network

The structure of a basic neural network is shown in Figure 3, where

- n_l, n_x, n_y: number of neurons/nodes in layer l, number of input variables, and number of output values respectively.
- $z_j^{[k]} = w_j^{[k]} a^{[k-1]} + b_j^{[k]}$: each node computes a linear combination using its weight, bias, and input value from its previous layer.
- $a_j^{[k]} = g(z_j^{[k]})$: then the node runs an activation function $g(z)$ onto the linear combination value to add nonlinearity and outputs the value to the next layer.

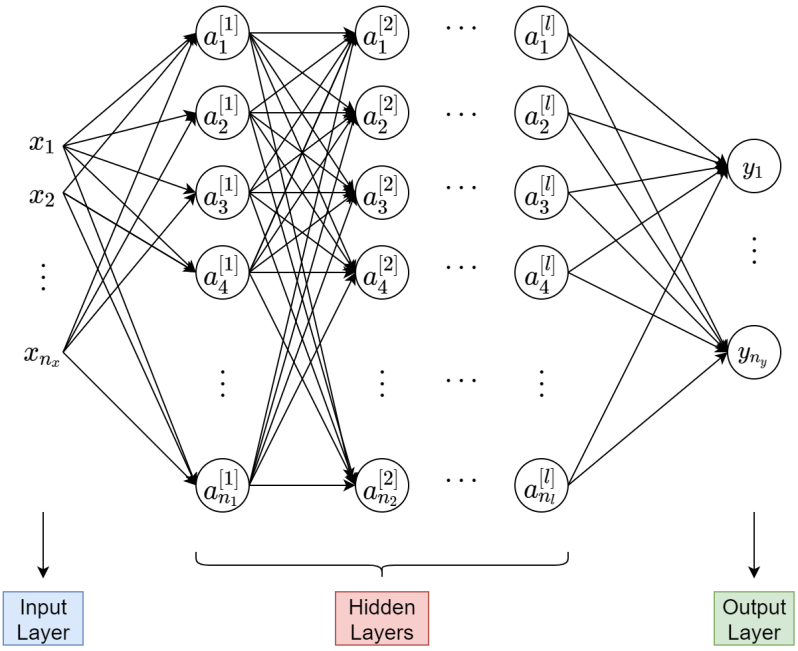

Figure 3. Neural network architecture

The training of a neural network consists of two major computation steps: forward and backward propagation (See Appendix for brief description of mathematical details). Forward propagation generates the predicted output values from the neural network, while backward propagation tries to minimize the differences between the actual and predicted output values (i.e. minimize the loss function) and updates the weights and biases values backwards, usually using gradient descent methods.

Bayesian neural network (BNN) is a neural network trained with Bayesian inference. Unlike a frequentist neural network which may suffer from overfitting and thus requires a dataset as broad as possible, BNN accounts for uncertainty and therefore allows flexibility in data availability. Regardless of the seemingly complicated structure of any neural network, we can always think of it as a blackbox function that transforms the inputs to the target outputs. With this intuition, we can always place a prior on the weight, define the likelihood based on the blackbox function, and generates posterior from prior and likelihood (Wang and Yeung, 2016), that is:

$$
\begin{aligned}
&W \sim N(\mu_W, \Sigma_W) \\
&y_i \sim N(f_W(x_i), \Sigma_y) \\
&p(W|D) \propto p(W) \prod_i p(y_i|x_i, W)
\end{aligned}
$$

assuming that the weight matrix W includes both weights w and bias b.

There are many advanced methods in BNN, and they generally center around the idea of probabilistic graphical models (PGM). For example, in latent Dirichlet allocation (LDA), one can impose a Dirichlet prior $w_i \sim \text{Dirichlet}(\alpha)$ and sample topics from a Multinomial distribution $z_{ij} \sim \text{Multinomial}(w_i)$, as discussed in Wang and Yeung (2016)'s paper. In this project, we will focus on the most basic Gaussian setting as illustrated above.

As mentioned in Chapter 3.1, VI is more favorable than MCMC in Bayesian deep learning. First, we define our variational posterior as (Blundell et al., 2015):

$$\mathbf{w} = \mu + \ln(1 + e^{\rho}) \circ \epsilon$$

where

- $\epsilon \sim N(0, I)$: a Gaussian noise (I: identity matrix).
- μ: assumed mean of variational posterior of $\mathbf{w}$.
- $\sigma = \ln(1 + e^{\rho})$: assumed standard deviation of variational posterior of $\mathbf{w}$. This transformation ensures that $\sigma \geq 0$.

We put the two variational posterior parameters that we aim to optimize inside a tuple $\theta = (\mu, \rho)$. Then, similar to ELBO in Chapter 3.1, we define a function of $\mathbf{w}$

and θ to be maximized (Blundell et al., 2015):

$$f(\mathbf{w}, \theta) = \ln q(\mathbf{w}|\theta) - \ln p(\mathbf{w}p(D|\mathbf{w}))$$

Then, we calculate the gradients:

$$\Delta_\mu = \frac{\partial f}{\partial \mathbf{w}} + \frac{\partial f}{\partial \mu}$$
$$\Delta_\rho = \frac{\partial f}{\partial \mathbf{w}} \frac{\epsilon}{1 + e^{-\rho}} + \frac{\partial f}{\partial \rho}$$

and we run gradient descent on the variational parameters just like in the original neural network:

$$\mu \leftarrow \mu - \alpha \Delta_\mu$$
$$\rho \leftarrow \rho - \alpha \Delta_\rho$$

Long Short-Term Memory

Long Short-Term Memory (LSTM) is the most widely used variation of Recurrent Neural Networks (RNN) for sequential modeling tasks. As an example of RNN architecture, Figure 3 shows the general structure of a many-to-many RNN.

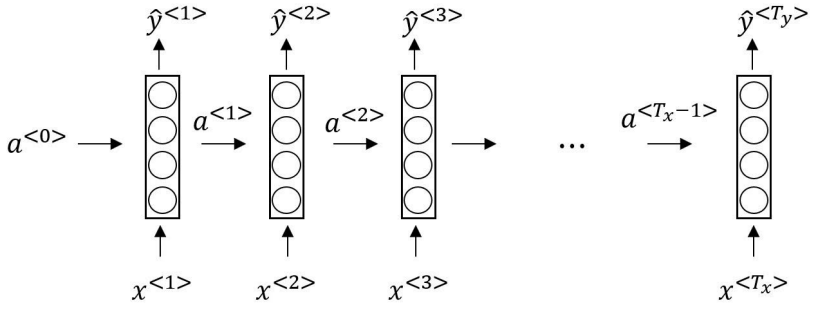

Figure 4. Example: many-to-many RNN

The architecture of an LSTM unit is (Hochreiter and Schmidhuber, 1997):

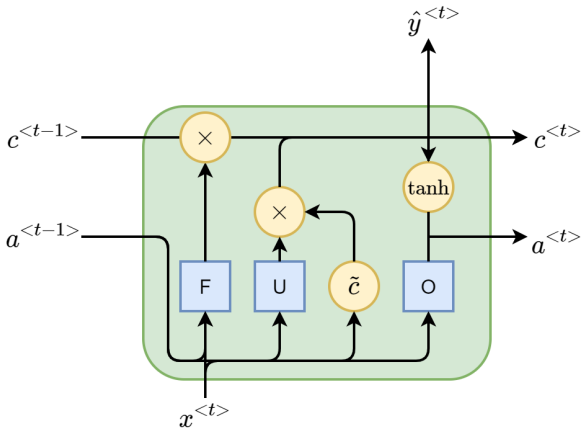

Figure 5. LSTM unit

The unit takes 3 inputs (input vector at current step $x^{<t>}$, activation value from last step $a^{<t-1>}$, and candidate value from last step $c^{<t-1>}$) and generates 3 outputs (predicted value at current step $\hat{y}^{<t>}$, activation value for next step $a^{<t>}$, and candidate value for next step $c^{<t>}$). There are 3 gates inside an LSTM unit:

- F gate (Forget): determines whether to forget the previous cell
- U gate (Update): determines whether to update the computation with the candidate
- O gate (Output): computes the normal activation as an output

The forward propagation process is (Hochreiter and Schmidhuber, 1997):

1. Compute F gate: $\Gamma_f = \sigma(W_f[a^{<t-1>}; x^{<t>}] + b_f)$
2. Compute U gate: $\Gamma_u = \sigma(W_u[a^{<t-1>}; x^{<t>}] + b_u)$
3. Compute O gate: $\Gamma_o = \sigma(W_o[a^{<t-1>}; x^{<t>}] + b_o)$
4. Compute candidate: $\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}; x^{<t>}] + b_c)$
5. Compute memory cell: $c^{<t>} = \Gamma_u \tilde{c}^{<t>} + \Gamma_f c^{<t-1>}$
6. Output activation for the next unit: $a^{<t>} = \Gamma_o \cdot \tanh c^{<t>}$

where

- $x^{<t>} \in \mathbb{R}^d$: input vector.
- $a^{<t-1>} \in \mathbb{R}^h$: output vector from previous step.
- $W_i \in \mathbb{R}^{h \times (d+h)}$ and $b_i \in \mathbb{R}^h$: the weights and biases for gate i respectively.
- $\tanh(z)$ and $\sigma(z)$: hyperbolic tangent and sigmoid function respectively.

- Semicolon ; means the two vectors are joined vertically. That is, $W_i[a^{<t-1>}; x^{<t>}] = W_{i,1:h}a^{<t-1>} + W_{i,(h+1):(h+d)}x^{<t>}$.

The Bayesian integration for LSTM is the same as for BNN.

Random Forest

Random Forest is an ensemble method of decision trees. A decision tree looks like the following:

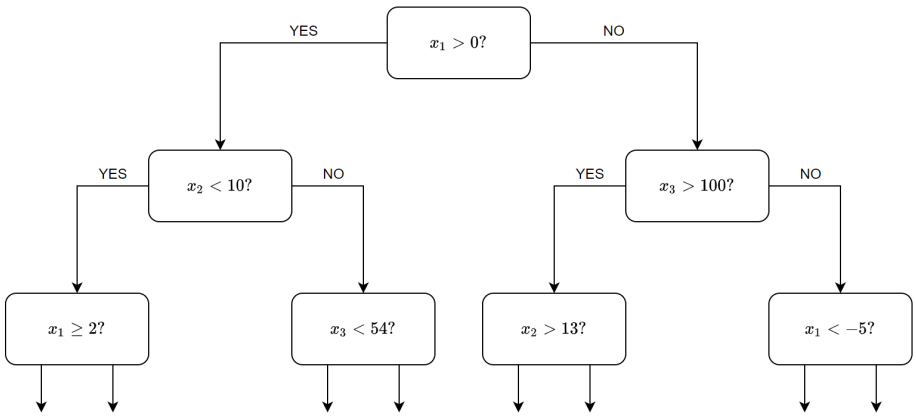

Figure 6. Example of a decision tree

The decision tree eventually divides the original data points into separate clusters based on its features. Each tree is trained individually on random samples by reducing the Gini impurity (a measure of incorrect labeling frequency) of each node along the tree till the last layer:

$$I_G(p) = 1 - \sum_{i=1}^{J} p_i^2$$

where J is the number of feasible classes at the node. A random forest bags a bunch of decision trees together (i.e. bagging) and averages the predictions from all trees as the output, mathematically expressed as (Liaw et al., 2002):

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x^*)$$

where B is total number of trees in the forest and $f_b(\cdot)$ represents the bth tree.

XGBoost

Unlike random forest which belongs to the bagging family, XGBoost belongs to the boosting family. After learning the first tree, we carry that knowledge to the learning of the next tree, and we continue this additive training:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

where t is the current time step and $f_k(x_i)$ is the kth tree. The real loss function that XGBoost aims to minimize is (Chen and Guestrin, 2016):

$$\sum_{i=1}^{m} \left(g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t)$$

where

- m: total number of samples
- $g_i = \frac{\partial \mathcal{L}}{\partial \hat{y}_i^{(t-1)}}$: first-order gradient (where $\mathcal{L}(y_i, \hat{y}_i^{(t-1)})$ is the original loss function that can be customized)
- $h_i = \frac{\partial^2 \mathcal{L}}{\partial \hat{y}_i^{(t-1)^2}}$: second-order gradient
- $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$: regularization term to prevent overfitting, where T is the number of leaves and w_j is the scores of jth leaf.

As previously mentioned, to add some Bayesian aspect to decision tree ensembles methods, we will apply Bayesian optimization to the hyperparameter tuning process of these methods.

Bayesian Predictive Synthesis

Bayesian Predictive Synthesis (BPS) is a brand new, pure Bayesian forecasting method created by McAlinn and West (2019). Though they have already developed the multivariate BPS, this project strictly focuses on the univariate BPS. Everything originates from the agent opinion analysis theory (McAlinn and West, 2019):

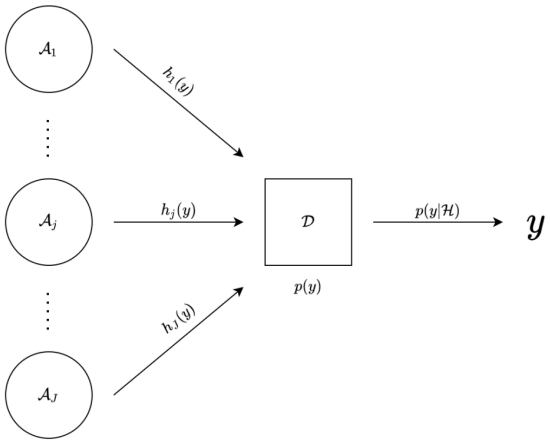

Figure 7. Bayesian predictive synthesis

where individual agents $\mathcal{A}_j$ $(j = 1 : J)$ inform a Bayesian decision maker $\mathcal{D}$ of their individual inferences $h_j(y)$ on the issue y. With the collection of these inferences $\mathcal{H} = \{h_1(\cdot), \cdots, h_J(\cdot)\}$, the decision maker $\mathcal{D}$ will generate a posterior $p(y|\mathcal{H})$ as the final prediction for y. More specifically,

$$p(y|\mathcal{H}) = \int_{\mathbf{x}} \alpha(y|\mathbf{x}) \prod_{j=1}^{J} h_j(x_j) d\mathbf{x}$$

where

- $\mathbf{x}$: latent agent state, a $1 \times J$ latent vector of dummy variables.
- $\alpha(y|\mathbf{x})$: calibration function, a probability density function for $y|x$.

In a temporal setting, at each time step t, $\mathcal{D}$ receives $\mathcal{H}_{t+1}$ from agents and attempts to forecast y_{t+1} based on $\{y_{1:t}, \mathcal{H}_{1:t+1}\}$. What is unique about BPS is that over time, $\mathcal{D}$ will learn more and more about the agents themselves based on the data received from them, thus leading to the addition of a time-varying parameter vector $\mathbf{\Phi}_t$ dependent on $\{y_{1:t-1}, \mathcal{H}_{1:t-1}\}$ into the original BPS:

$$p(y|\mathbf{\Phi}_t, \mathcal{H}_t) = \int \alpha_t(y_t|\mathbf{x}_t, \mathbf{\Phi}_t) \prod_{j=1}^{J} h_{tj}(x_{tj}) d\mathbf{x}_t$$

Though it looks confusing, the core idea is no different from all the previous discussions on Bayesian inference: $\prod_{j=1}^{J} h_{tj}(x_{tj})$ is the prior, and $\alpha_t(y_t|\mathbf{x}_t, \mathbf{\Phi}_t)$ is the likelihood.

For prior, each latent variable x_{tj} is an independent latent draw from $h_{tj}(\cdot)$. Therefore,

$$p(\mathbf{x}_t|\mathbf{\Phi}_t, y_{1:t-1}, \mathcal{H}_{1:t}) \equiv p(\mathbf{x}_t|\mathcal{H}_t) = \prod_{j=1}^{J} h_{tj}(x_{tj})$$

Also, McAlinn and West (2019) used the standard dynamic linear model (DLM) for the calibration function:

$$\begin{aligned} y_t &= \mathbf{F}_t^T \boldsymbol{\theta}_t + \nu_t, \quad \nu_t \sim N(0, v_t) \\ \boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} + \boldsymbol{\omega}_t, \quad \boldsymbol{\omega}_t \sim N(0, v_t \mathbf{W}_t) \end{aligned}$$

where $\mathbf{F}_t = (1, \mathbf{x}_t^T)^T$, $\boldsymbol{\theta}_t = (\theta_{t0}, \cdots, \theta_{tJ})^T$.

Due to the evolution dynamics in DLM, we can directly represent our time-varying parameter vector as $\mathbf{\Phi}_t = (\boldsymbol{\theta}_t, v_t)$. Since both $\mathbf{x}_t$ and $\mathbf{\Phi}_t$ are defined, we can write the calibration function:

$$\alpha_t(y_t|\mathbf{x}_t, \mathbf{\Phi}_t) = N(y_t|\mathbf{F}_t^T \boldsymbol{\theta}_t, v_t)$$

For posterior inference, McAlinn and West (2019) constructed a two-component Gibbs sampler:

1. Draw parameters from $p(\mathbf{\Phi}_{1:t}|\mathbf{x}_{1:t}, y_{1:t})$ by forward filtering backward sampling (FFBS) algorithm.
2. Draw agent states from $p(\mathbf{x}_{1:t}|\mathbf{\Phi}_{1:t}, y_{1:t}, \mathcal{H}_{1:t})$ by MH algorithm, where

$$p(\mathbf{x}_t|\mathbf{\Phi}_t, y_t, \mathcal{H}_t) \propto \alpha_t(y_t|\mathbf{x}_t, \mathbf{\Phi}_t) p(\mathbf{x}_t|\mathcal{H}_t)$$

Further technical details of this MCMC approach are discussed in their paper.

4. Experiment

GPR, BNN, LSTM, Random Forest, and XGBoost were implemented with Python, while SVR and BPS were implemented with Julia. The train-test split ratio is 0.8:0.2 (i.e. the train set is 2011/06/01 - 2019/05/31, and the test set is 2019/06/01 -

2021/06/01). Most models were implemented and tested successfully, but a technical issue occurred during the testing of BPS. My laptop crashed and the CPU became overheated due to insufficient computational power. Therefore, I reduced the data size by using monthly average prices in place of daily prices. This reduced the data length from around 2400 to only around 120.

The GPR model has the kernel structure: $k_{\text{GPR}} = k_{\text{noise}} + k_{\text{const}} \times k_{\text{RBF}}$, where k_{noise} is a white noise kernel with a noise level of 9 fluctuating in the interval [1,25], k_{const} is a constant kernel with a constant value equivalent to the mean of the time series, and k_{RBF} is an RBF kernel with a length-scale of 10. The ϵ-SVR model also uses the RBF kernel with a length-scale scaled by the size of each time series, and it has a tolerance hyperparameter $C = 1000$ and $\epsilon = 10^{-5}$. For both regression models, the CI multipler was set to 0.5.

The BNN consists of 3 dense layers with 8, 16, 8 neurons respectively. Due to the lack of functionality of my laptop, I was not able to run more layers and more neurons. Nevertheless, a BNN with a more complex structure (such as a shape of $16 \times 32 \times 16$) did not lead to better results. For the purpose of this research, this structure is sufficient. The LSTM neural network consists of 1 LSTM layer with 1 input unit and 16 output units. For both neural networks, I chose ReLU (Rectified Linear Unit) as the activation function for all neurons, MSE (Mean Squared Error) as the loss function, and Adam (Kingma and Ba, 2017) as the optimizer. The total iterations was 2000 and the batch size was 32. In addition, the CI multipler was set to 3. This seems a bit too large as it returns a 99% credible interval, but the variances of predictions were quite small to capture the actual values due to the nature of variational inference method for the network weights.

The Random Forest and XGBoost models went through Bayesian optimization. The chosen hyperparameters for Random Forest are: max_depth (in [3,10]), min_sam -ples_leaf (in [1,4]), min_samples_split (in [2,10]), and n_estimators (in [100,200]). The chosen hyperparameters for XGBoost are: max_depth (in [3,10]), gamma (in [0,1]), colsample_bytree (in [0.1,0.9]), eta (in [0.1,0.5]), and subsample (in [0.5,1]). Please refer to Liaw et al. (2002) and Chen and Guestrin (2016) or check the official

documentations for Scikit-learn (Pedregosa et al., 2011) and XGBoost for the details of these hyperparameters.

The BPS model consists of 4 agents, each with a prior based on a Gaussian distribution with a mean of the previous value in the time series and a variance dependent on the total size of the time series. The burn-in was set to 1000, the number of MCMC iterations was set to 2000, and the CI multiplier was set to 2 for a 95% credible interval.

5. Results

The results are separated in two streams: Credible Interval (CI) metric and Root Mean Squared Error (RMSE) metric. Due to the tolerance of uncertainty in the final outputs in Bayesian methods, we define the accuracy metric as the share of the actual values that lie within the predicted credible intervals from the Bayesian models:

$$\text{CI Inclusion} \equiv P(\text{UCB} \geq \text{value} \geq \text{LCB})$$

where UCB is the upper credible bound and LCB is the lower credible bound. The P here represents frequency. Since Random Forest and XGBoost have deterministic outputs, their results are not included in the credible interval section. Thus, we use the RMSE metric in addition, in which we use the posterior mean as our model prediction and compare it with the actual values:

$$\text{RMSE} \equiv \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}$$

where m is the size of the test set.

Table 3 and Table 4 are the forecasting results. Forecasting plots are provided in Appendix instead due to the huge quantity of them. For consistency of plots, the forecast posterior mean and variance of SVR and BPS were exported into csv files and plotted in Python. The results from BPS are separated from other models due to different underlying datasets.

Table 3: CI Inclusion Accuracy Results (in %)

	GPR	SVR	BNN	LSTM	BPS
Nikkei225	**99.58**	69.05	29.07	67.01	**100.00**
NASDAQ	**63.62**	28.23	17.84	48.69	**100.00**
SSEC	**99.81**	99.40	94.19	97.30	**99.46**
HSI	**99.79**	99.38	77.00	77.82	**99.89**

Two findings can be addressed in Table 3 through model comparison and dataset comparison. First, pure Bayesian models seem to dominate the CI inclusion accuracy for all 4 datasets. For daily data, GPR has an exceptional performance compared to SVR, BNN and LSTM even though the kernel structure was not complicated. For monthly average data, BPS showed exceptional performance. However, this finding is disputable because the size of credible intervals are dependent on the posterior variance. According to Figure 10-13 and Figure 16, the span of the credible intervals of GPR and BPS were much larger than those of SVR, BNN and LSTM, indicating a higher predicted variance from GPR and BPS. For this reason, I included the RMSE results in the analysis as well. In addition to the highest CI inclusion accuracy of pure Bayesian methods, the two regression models (GPR and SVR) performed better than the two neural networks (BNN and LSTM) on all the 4 datasets. This could indicate that sophisticated model structures do not necessarily perform better than simple machine learning methods in time series forecasting, particularly when the time series is non-stationary. Despite the relatively simple architecture of the BNN and LSTM in this project, the structures of the GPR and SVR were also not complicated. Actually, more complicated BNN structures were tested as well through adding hidden layers, adding neurons on each layer, using different activation functions, etc. However, they resulted in a much worse performance than the current BNN structure (3 hidden dense layers with 8, 16, 8 neurons respectively). That is the reason why I did not use them in this paper. As Shen et al. (2012) implied in their result section, complicated structures do not necessarily mean advanced performance for forecasting, and the simple GPR and BPS in this project proved it. It is worth noting that according to their findings, BNN should have a better performance than LSTM for forecasting if

the hyperparameters are tuned properly, which contradicts the results from Figure 10 and 11. Therefore, improvements on hyperparameter choices and implementations are necessary for further analysis of the performance of neural networks in time series forecasting.

Second, the performance of the 4 models on Nikkei225 and NASDAQ is sufficiently worse than on SSEC and HSI in terms of CI inlusion accuracy. This result was surprising to me at first, but after looking at the plots and the data ranges, the reason seems clear. As indicated earlier in Table 1, the price range of Nikkei225 and NASDAQ increased by around 3 times and 6 times respectively over the decades. As shown in Table 2, Nikkei225 and NASDAQ price data are more non-stationary than SSEC and HSI. Such dramatic, non-stationary increases in the prices of Nikkei225 and NASDAQ are quite visible in Figure 10 and Figure 11 in Appendix. The price of Nikkei225 has at least fluctuated a bit during 2011-2021, while the price of NASDAQ just kept increasing from 2011 to 2021 except when the pandemic took place in the United States. On the other hand, both SSEC and HSI has been through fluctuations during 2011-2019, and thus the machine learning algorithms have experienced fluctuations during the training process. The COVID-19 pandemic caused a fluctuation in the stock indices as shown in Figure 1, and algorithms that encountered fluctuations in training handled fluctuations better in testing.

Table 4: RMSE Results

	GPR	SVR	BNN	LSTM	RF	XGBoost	BPS
Nikkei225	**599.22**	4887.60	3506.62	2368.35	2464.90	2445.03	**109.01**
NASDAQ	**872.53**	4932.07	3673.23	2890.25	3107.07	3118.09	**54.82**
SSEC	38.29	**35.63**	108.30	92.66	44.43	43.58	**18.56**
HSI	353.16	**341.44**	1001.96	778.42	388.46	385.76	**108.53**

The scale of RMSE depends on the scale of the original data points. As shown in the plots in Appendix, the scales of different stock indices are very different from each other, resulting in the different RMSE scales. Based on the RMSE results

and the forecasting plots in Appendix, GPR has the best accuracy in Nikkei225 and NASDAQ, while SVR has the best accuracy in SSEC and HSI. Interestingly, SVR has the worst possible RMSE results in Nikkei225 and NASDAQ. According to Figure 10-15, SVR actually forecasts a decreasing pattern rather than increasing for Nikkei225 and NASDAQ, while BNN, Random Forest, and XGBoost stayed at a maximal linear boundary when Nikkei225 and NASDAQ increased too much. Only GPR and LSTM tried their best to capture the upward trend of stock indices after the COVID-19 pandemic even tho such dramatic increase has not taken place during 2011-2019. The RMSE results provide evidence for the first finding from Table 3, that pure Bayesian methods appear to have intrinsic advantage in forecasting.

It is worth noting that decision tree ensembles models are not originally designed for time series forecasting. Nevertheless, after Bayesian optimization for hyperparameter tuning, they actually show competitive performance to SVR and LSTM, even when LSTM is designed for sequential modeling tasks. The tuned hyperparameters are summarized in Table 5 and Table 6. Random Forest has a clear distinction of hyperparameters between the increasing indices (Nikkei225 and NASDAQ) and the fluctuating indices (SSEC and HSI), while XGBoost recognizes NASDAQ as the unique index with a much higher maximal depth and an extremely low γ. Recall in Chapter 3.2 that γ is a regularization hyperparamter. Such an extremely low γ indicates the unnecessity of regularization for learning NASDAQ.

Last but not the least, the most exceptional performance is presented by BPS. The CI inclusion accuracy of BPS is almost 100% for all four datasets, and the RMSE results are significantly lower than the second best models, GPR and SVR. One possible reason behind such an extraordinary performance is the prior choice of the agents. The prior values of all 4 agents are randomly drawn from a Gaussian distribution of the previous time series value. According to the comparison between Figure 10-15 and Figure 16, the daily prices were much noisier than the monthly average prices. Converting daily prices into monthly average prices reduced the volatility of the time series and thus might have enhanced the autocorrelation within the time series. Therefore, this choice of agent priors played a significant role in the extraordinary performance of BPS. Unless the underlying datasets and conditions are the same for

BPS and the other models, no decisive assessment can be made about the relative performance of BPS.

6. Conclusion

This research investigated Bayesian methods in Machine Learning through stock forecasting. 7 models (GPR, SVR, BNN, LSTM, Random Forest, XGBoost, BPS) were examined in theoretical details, implemented on the price data of 4 stock indices (Nikkei 225, NASDAQ, SSEC and HSI) during 2011-2019, and evaluated on the price data during the pandemic period 2019-2021. Four findings can be addressed from this research. First, a sufficient level of computational power is necessary for the successful applications of Bayesian methods. Second, pure Bayesian methods (GPR and BPS) perform exceptionally well in stock index forecasting among all 7 models. GPR and BPS excel in both CI inclusion accuracy and RMSE criteria, but the high variance in the credible intervals of them should be taken into account when assessing their full performance. Third, complexity in the model structure does not necessarily improve the performance in stock index forecasting. GPR and SVR generated very promising results with simple structures, while BNN and LSTM with neural network structures did not. Also, complicating the structure of BNN only generated worse performance. Fourth, Bayesian optimization for hyperparameter tuning is highly beneficial, as shown in the competitive performance of the decision tree ensembles models (Random Forest and XGBoost) that were not originally designed for forecasting tasks.

As George E. P. Box stated, "All models are wrong, but some are useful." (Box, 1976) There are many more advanced Bayesian methods and forecasting models out there, such as the classical ARIMA and GARCH models, Kalman filter, state-space models, etc. In the recent years, reinforcement learning trading bots seem to capture the attention of researchers and practitioners (Charpentier et al., 2021), and Bayesian reinforcement learning is particularly interesting for exploring uncertainties, reacting to randomness, and competing against fellow trade bots (Ghavamzadeh et al., 2016). Also, meta learning is becoming popular in the recent years due to its advantage in

hyperparameter tuning. Bayesian meta learning is a new branch that might open possibilities (Lemke et al., 2015). It is still amazing that one simple theorem from around 200 years ago (Bayes, 1763) might become a key to uncovering the future uncertainties and stimulating the advancement towards Artificial Intelligence.

Appendix

Optimal hyperparameters for decision tree ensembles

The following two tables summarize the optimal hyperparameters for Random Forest and XGBoost for different stock index data. A brief description for each is provided below (Pedregosa et al., 2011) (Chen and Guestrin, 2016).

Table 5: Optimal Hyperparameters for Random Forest

	max_depth	min_sample_leaf	min_sample_split	n_estimators
Nikkei225	10	1	2	100
NASDAQ	10	1	2	100
SSEC	10	1	5	190
HSI	10	1	5	139

Table 6: Optimal Hyperparameters for XGBoost

	max_depth	gamma	colsample_bytree	eta	subsample
Nikkei225	3	0.37	0.90	0.1	0.8
NASDAQ	6	0.06	0.90	0.1	0.8
SSEC	3	1.00	0.90	0.1	0.8
HSI	3	0.64	0.83	0.1	0.8

- max_depth: maximal depth of a tree.
- min_sample_leaf: minimal sample leaf size.
- min_sample_split: minimal sample splits.
- n_estimators: number of trees before aggregation.
- gamma: minimal loss reduction on a leaf node of a tree.
- colsample_bytree: subsample ratio of columns when constructing each tree.
- eta: step size shrinkage for regularization.
- subsample: subsample ratio of the training instances.

Neural network propagation process

The following two figures indicate the calculation process of a basic neural network. In the forward propagation, each neuron takes the input from the previous layer (x if from the very first input layer, $a^{[l-1]}$ otherwise), generates a linear combination z of it with its own weight w and bias term b, activates it with a non-linear activation function $g(z)$, and outputs it $a^{[l]}$ to the next layer $l+1$. At the final output layer L, the computed output value will be compared with the actual output value through a loss function $\mathcal{L}(y-\hat{y})$.

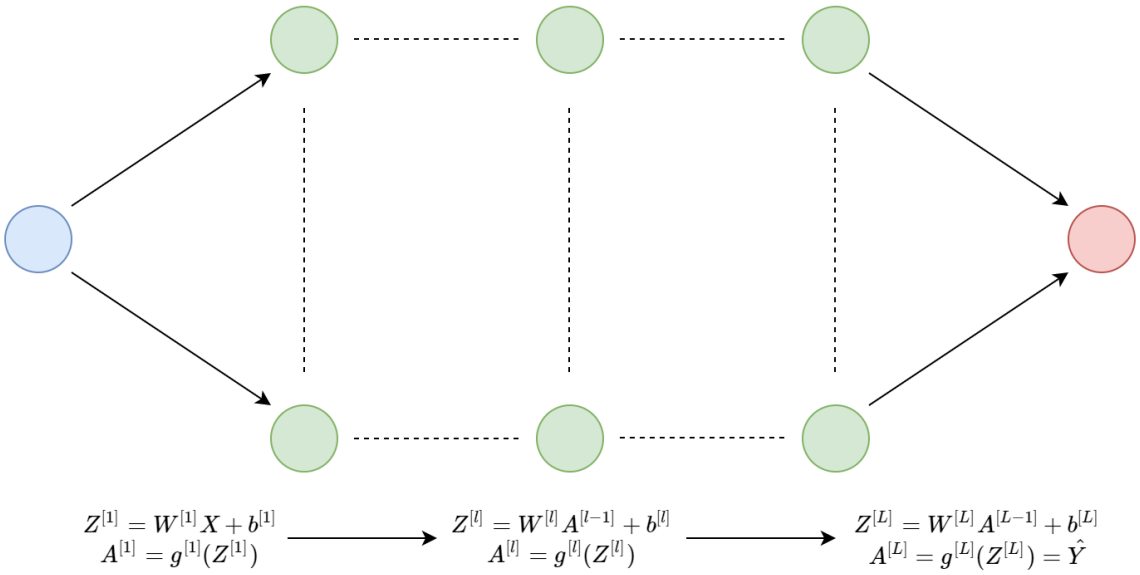

Figure 8. Forward propagation

In the backward propagation, the loss function will be differentiated with respect to the activation function at each neuron on the last hidden layer first to generate the gradient with respect to the activation value $\frac{\partial \mathcal{L}}{\partial a^{[L]}}$. Based on the chain rule, the gradient with respect to the linear combination value can be computed as $\frac{\partial \mathcal{L}}{\partial z^{[L]}} = \frac{\partial \mathcal{L}}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}}$, and the gradient with respect to the weight and bias term can be further calculated. With the help of chain rule, a chain of gradients is generated as the calculation propagates backwards till the very first input layer. Then, on each neuron, gradient descent is performed to find the optimal weight and bias that minimizes the

loss function.

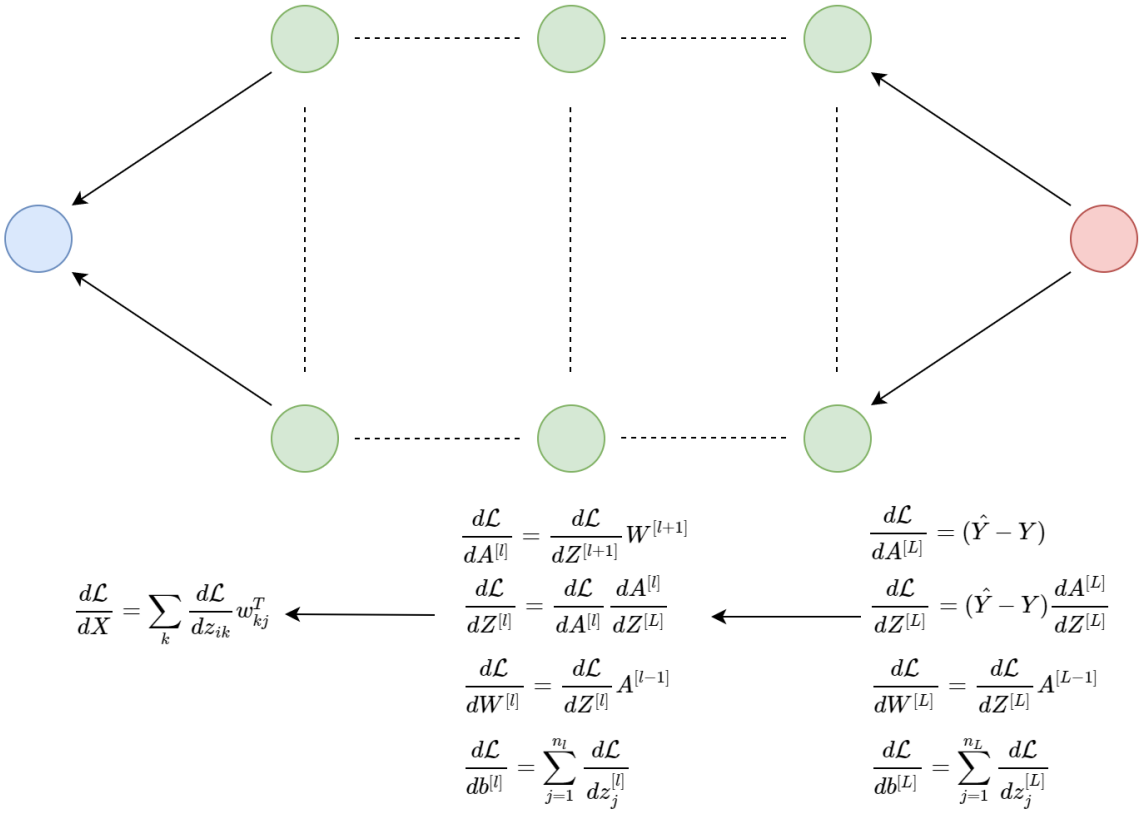

Figure 9. Backward propagation

Forecasting plots

Figure 10-13 are the forecasting plots of the 4 stock indices from GPR, SVR, BNN, and LSTM in order. Figure 14-15 are the forecasting plots from Random Forest and XGBoost models. Figure 16 is the forecasting plots from BPS. The black curve represents the actual values, the red curve represents the predicted mean values, and the green span represents the credible intervals for the predicted values.

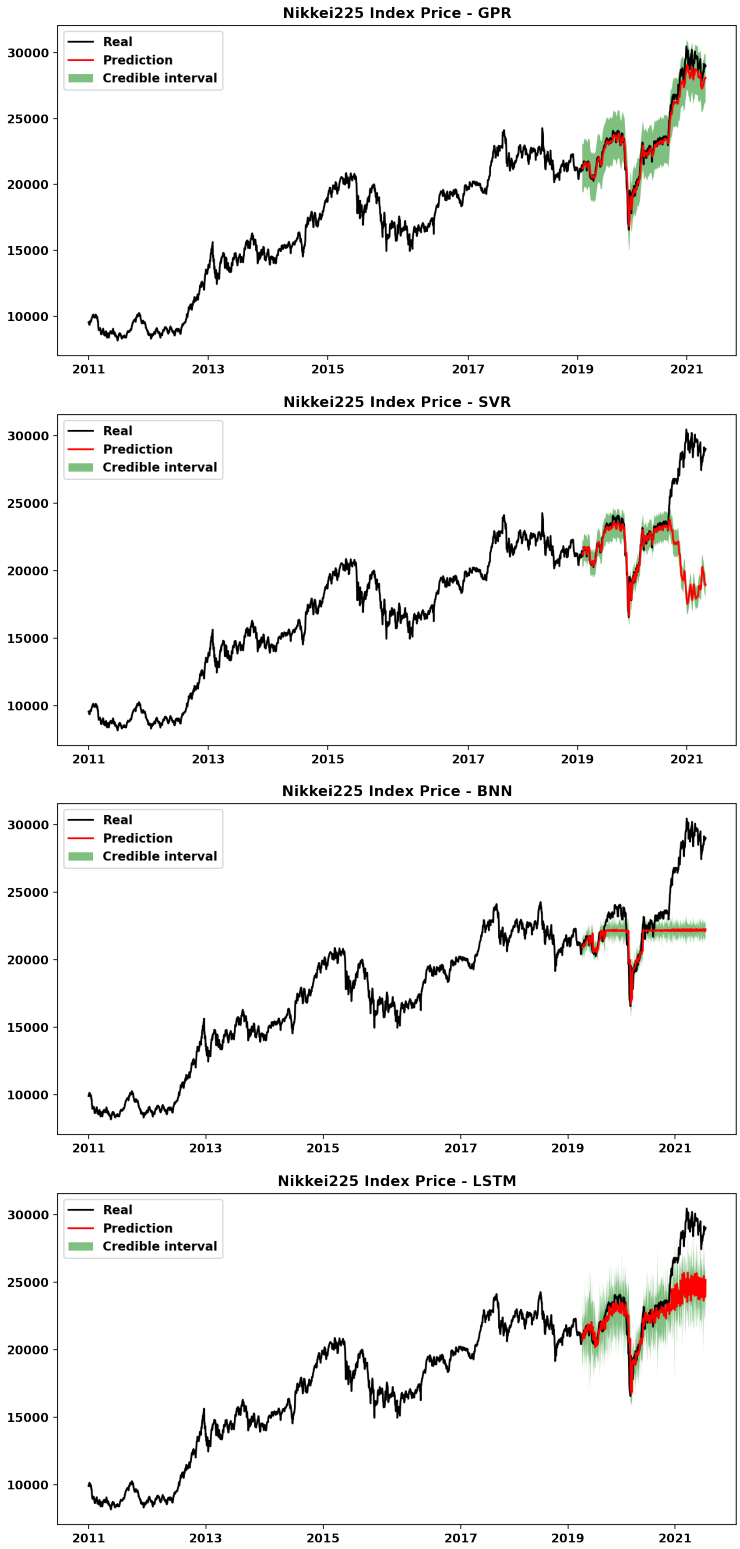

Figure 10. Nikkei225 forecasting plots

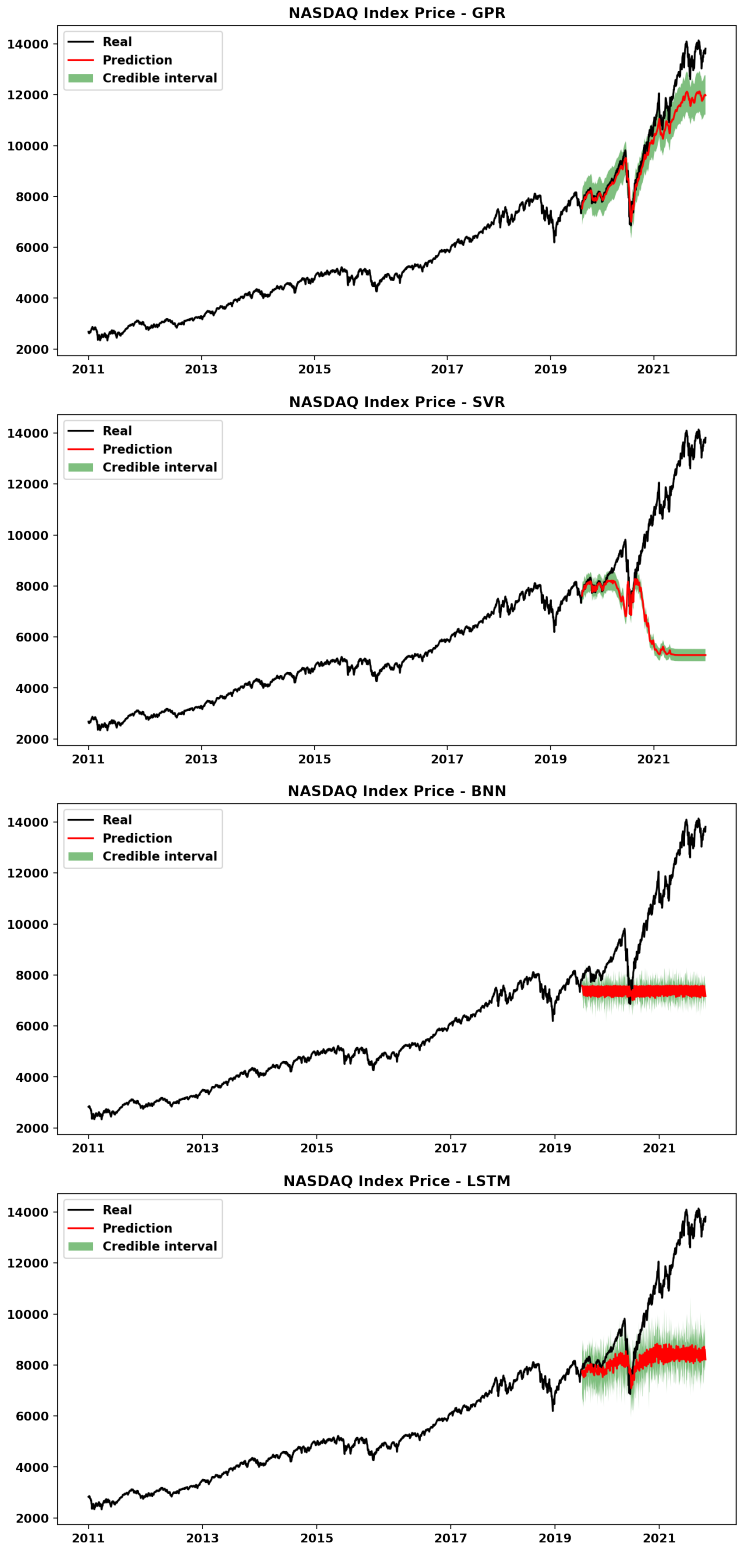

Figure 11. NASDAQ forecasting plots

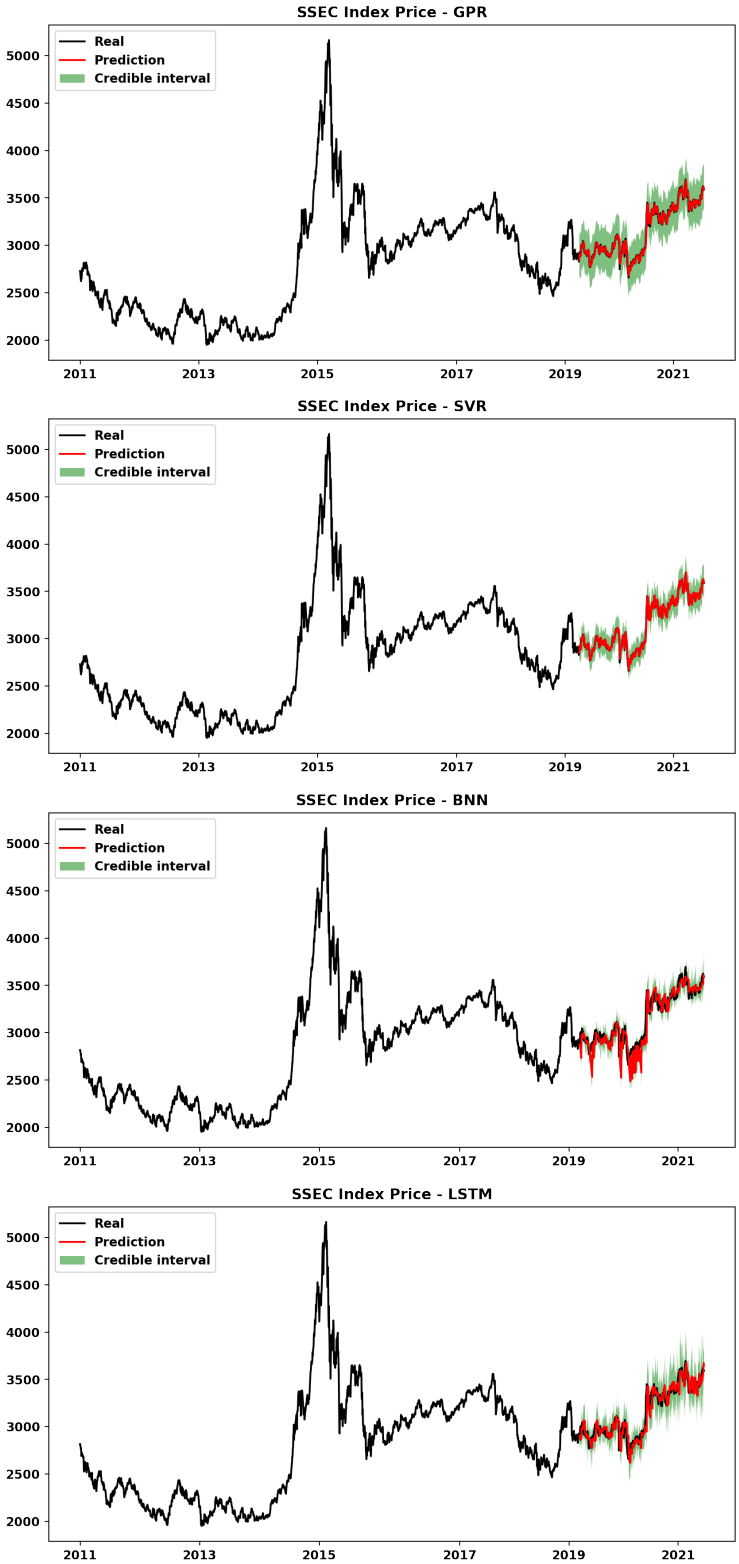

Figure 12. SSEC forecasting plots

Figure 13. HSI forecasting plots

Figure 14. Random Forest forecasting plots

Figure 15. XGBoost forecasting plots

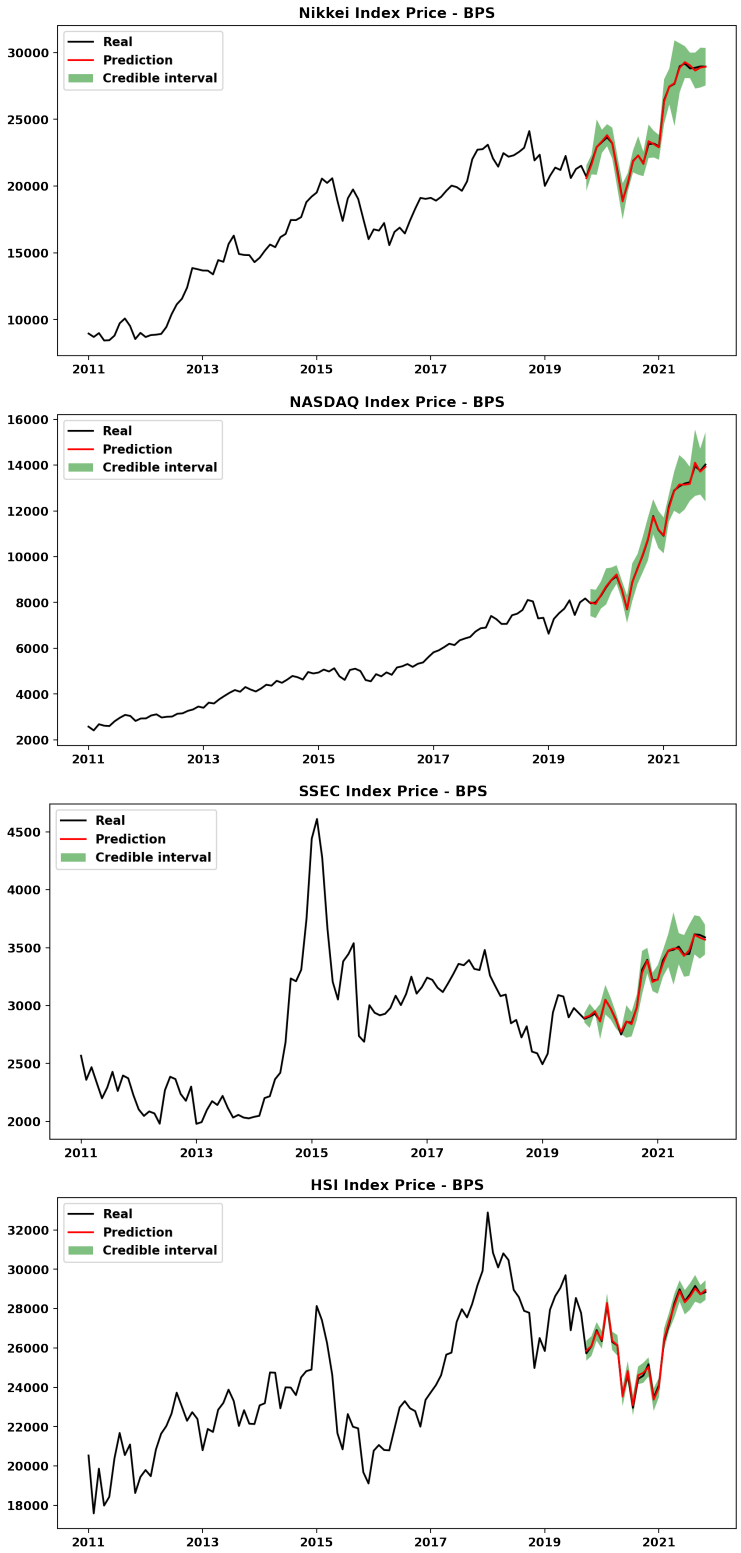

Figure 16. BPS forecasting plots

References

Baker, S. R., Bloom, N., Davis, S. J., Kost, K., Sammon, M., and Viratyosin, T. (2020). The unprecedented stock market reaction to covid-19. *The Review of Asset Pricing Studies*, 10(4):742–758.

Bayes, T. (1991). An essay towards solving a problem in the doctrine of chances. 1763. *MD computing: computers in medical practice*, 8(3):157–171.

Betancourt, M. (2017). Stan conference 2017.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR.

Box, G. E. (1976). Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799.

Charpentier, A., Elie, R., and Remlinger, C. (2021). Reinforcement learning in economics and finance. *Computational Economics*, pages 1–38.

Chen, S.-W. et al. (2008). Non-stationarity and non-linearity in stock prices: Evidence from the oecd countries. *Economics Bulletin*, 3(11):1–11.

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

Chib, S. and Greenberg, E. (1995). Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A., Vapnik, V., et al. (1997). Support vector regression machines. *Advances in neural information processing systems*, 9:155–161.

Gelfand, A. E. and Smith, A. F. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85(410):398–409.

Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A. (2016). Bayesian reinforcement learning: A survey. *arXiv preprint arXiv:1609.04436*.

Hamilton, J. D. (2020). *Time series analysis*. Princeton university press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623.

Jospin, L. V., Buntine, W., Boussaid, F., Laga, H., and Bennamoun, M. (2020). Hands-on bayesian neural networks–a tutorial for deep learning users. *arXiv preprint arXiv:2007.06823*.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Law, M. H. and Kwok, J. T.-Y. (2001). Bayesian support vector regression. In *AISTATS*, page 239C244. Citeseer.

Lemke, C., Budka, M., and Gabrys, B. (2015). Meta learning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

McAlinn, K. and West, M. (2019). Dynamic Bayesian predictive synthesis in time series forecasting. *Journal of Econometrics*, 210(1):155–169.

Nicola, M., Alsafi, Z., Sohrabi, C., Kerwan, A., Al-Jabir, A., Iosifidis, C., Agha, M., and Agha, R. (2020). The socio-economic implications of the coronavirus and covid-19 pandemic: a review. *International journal of surgery*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N., and Aigrain, S. (2013). Gaussian processes for time-series modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1984):20110550.

Shen, S., Jiang, H., and Zhang, T. (2012). Stock market forecasting using machine learning algorithms. *Department of Electrical Engineering, Stanford University, Stanford, CA*, pages 1–5.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*.

van de Schoot, R., Depaoli, S., King, R., Kramer, B., Märtens, K., Tadesse, M. G., Vannucci, M., Gelman, A., Veen, D., Willemsen, J., et al. (2021). Bayesian statistics and modelling. *Nature Reviews Methods Primers*, 1(1):1–26.

Wang, H. and Yeung, D.-Y. (2016). Towards bayesian deep learning: A framework and some existing methods. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3395–3408.