The University of Hong Kong

Faculty of Engineering

Department of Computer Science

COMP7704

Dissertation

The Prediction of Financial Sequences based on EMD and Machine Learning

Submitted in partial fulfillment of the requirements for the admission to the degree of Master of Science in Computer Science

By
Jiajun Cai
UID: 3035561431

Supervisor: Dr. J.R. Zhang
Date of submission: 15/07/2020

ABSTRACT

Forecasting of financial time series is always a difficult and tempting topic over the world. It becomes extremely hard when no extra information can be utilized. To solve it, this dissertation proposes a framework based on Empirical Mode Decomposition (EMD) and machine learning models with the general idea of "decompose + recompose". We adopt five machine learning models including Prophet, DeepAR, Long Short-Term Memory Network (LSTM), support vector regression (SVR) and XGBoost model into our framework and apply the framework in the forecasting of S&P500 index, exchange rate of USDGBP and Brent Oil price. The experiment results demonstrate that: 1) the framework successfully improve the forecasting result with valuable practical reference; 2) the framework has strong robustness by employing model ensemble method.

DECLATRATION

I solemnly declare that this dissertation is completed independently under the guidance of my supervisor. Except for the content specifically quoted and cited, this paper does not contain any other individual or collective work that has been published or written. I have read the related rules of copyright and plagiarism and I am fully aware the corresponding consequences.

Signature: Cai Jiajun

Date: 15/07/2020

ACKNOWLEDGMENT

Firstly, I want to give my greatest gratitude to my supervisor, Dr. J.R. Zhang, who give me great freedom for my schedule of this dissertation and approve me to research this topic. Dr. Zhang always give me valuable advices to my paper.

Then, I am also grateful to my internship company, Huawei Noah's Ark Lab, and my internship colleagues. They provide me the luxurious experiment environment to verify my hypothesis and Dr. Shi give me the chance to have an acquaintance of time series forecasting in real industry.

Lastly, I want to thank my parents for giving me financial support during my postgraduate. Department of Computer Science at the University of Hong Kong is also gratefully acknowledged.

CONTENTS

1. INTRODUCTION

In financial market, most investment institutions and individual investors need to forecast underlying asset price to manage or appreciate their funds or property. According to the Efficient Market Hypothesis, all available information can be fully reflected by the current price so that normal approaches for predicting the accurate value of financial time series are hard to forecast well. However, if we decompose financial time series into serval components and get the predictions of each component, then compose these sub-predictions to the final result which is the prediction of original financial time series, it probably improves predictive performance than some non-decompose approaches.

This article proposes a framework for financial time series forecasting which includes three procedures, decomposition, prediction and combination. In the part of decomposition, because of the huge instability of financial series, which causes difficulties for many forecasting methods with presupposition that series are stationary such as Auto-regression algorithm, Empirical Mode Decomposition algorithm (EMD) (Huang et al., 1998) is adaptive to decompose the original time series into several stable subsequences. In the part of prediction and combination, multiple algorithms were adaptived to forecast each intrinsic mode functions (IMF), i.e. the components after decomposing by EMD algorithm, to get sub-predictions of each IMF. Finally, we build meta-learners for each component by stacking the outputs of each model and combine the outputs of meta-learners to form the ultima prediction of original financial time series. In this thesis, we use Long Short-Term Memory (LSTM) (Hochreiter &

Schmidhuber, 1997) algorithm, Support Vector Regression (SVR) (Smola & Schölkopf, 2004) algorithm, XGBoost (Chen & Guestrin, 2016), Prophet (Taylor & Letham, 2018) and DeepAR(Salinas, Flunkert, Gasthaus, & Januschowski, 2019) as prediction models.

In this paper, we focus on the prediction with no additional information involved like public sentiment but only the original price itself because that addition information is not predictable although some of them have an effect on their relative asset price. Therefore, the accuracy of prediction result may not be satisfied in real investment, but it is superior to those models with no decomposition and the individual model in above. Notwithstanding, the prediction of our model can reflect the future price when the market is not much changeable and becomes an important reference for investment operation.

The rest of this article is organized as follows. In section 2, we discuss an overview of the existing literature and some relative works. Section 3 describes the technique including models and decomposition method used in our framework, while section 4 presents our proposed framework. The next section describes the detailed experiment setting and compared models. Section 6, we present and analyze the experiment results and a conclusion of this paper was offered in section 7.

2. LITERATURE REVIEW

2.1 financial series forecasting

Financial series is various, for example, company's stock price series, stock index price series like Dow Jones Index and Nasdaq Composite Index, precious metals price series like gold price, commodity future price series like petroleum future price, the GDP yearly growth etc. To earn better investment income, investors and professional researchers need to forecast the future prices of financial series, thus, financial series forecasting becomes a significant topic for them.

2.1.1 statistic-based methods

There are several approaches in financial forecasting area. In the past, the major one is statistic-based methods , which include moving averages smoothing, autoregressive moving average model (ARMA), Holt-Winters' seasonal method, autoregressive conditional heteroskedasticity model (ARCH), generalized autoregressive conditional heteroskedasticity model (GARCH) (Bollerslev, 1986), vector autoregressive model (VAR) (Gordon, King, & Modigliani, 1982), etc.

Downs and Rocke (1983) adopted ARMA model to the problem of forecasting city budget variables. However, this method requires inputting stationary time series, which is not applicable for volatile data. In contract, Holt-Winters' seasonal method can accept the non-stationary data and captures the seasonal trend, for example, Agapie (1997) proposed an extension of the Holt-Winters's seasonal method to predict the economic cycles of GDP growth and the agricultural component of GDP. ARCH method specializes in volatility of time series with the supposition of independent identically distribution and was widely used in empirical analysis of financial engineering. Issler (1999) presented an empirical analysis of the return and conditional variance of Brazilian financial series based on ARCH method. Meanwhile, GARCH, an important variant method of ARCH proposed by Bollerslev in 1986, can simulate the volatility of time series when series requires high order of ARCH to do so. It was adopted to forecast stock market volatility of DAX, EOE and other three stock indices by Franses and Van Dijk (1996). For multiple financial time series, there are some linear interdependencies among them, and VAR method is used to capture these relationships by vector operation. Dhakal, Kandil, and Sharma (1993) researched the causality between the money supply share prices based on VAR method.

These methods have follows advantages: (i) lower computation cost compared to learning system approaches; (ii) the prediction result have good robustness, i.e. the result will not deviate far from the target especially when the data quality is not good enough; (iii) Outstanding interpretability. These methods have clear math formula which can interpret its meaning well. For instance, ARMA algorithm is composed by sum of weighted historical series and white noise. At the same time, the disadvantages are also obvious: (i) some complex patterns cannot be recognized due to its simple framework; (ii) some conditions of data need to be satisfied by some of these methods like autoregressive moving average algorithm (ARMA) which require stationary data.

2.1.2 learning-system-based methods

With the development of machine learning and computation ability, learning-system-based methods are widely adopted in the forecasting of financial time series, which include machine-learning-based methods like support vector machine (SVM) (Smola & Schölkopf, 2004), XGBoost (Chen & Guestrin, 2016) and neural-network-based methods like long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997), Prophet (Taylor & Letham, 2018), DeepAR (Salinas et al., 2019), etc.

SVR, regression form of SVM, is well fitted in small-scale time series dataset contained non-linear relationship. For example, a two-stage forecasting methods based on independent component analysis and SVR (regression model of SVM) was proposed to predict the daily Nikkei 225 opening cash index and TAIEX closing cash index (Lu, Lee, & Chiu, 2009). But its performance is declined when the scale of data becomes larger. XGBoost, a scalable tree boosting method, can solve this problem. Zhou, Li, Shi, and Qian (2019) proposed a XGBoost-based model to predict West Texas Intermediate (WTI) crude oil prices. However, numbers of hyperparameters need to be tuned to acquire good performance. With increasing computation power, it makes neural-network-based methods to gain enough training. In sequence forecasting, the major models are based on recurrent neural network. LSTM method is a gate-control algorithm based on recurrent neural network which is able to simulate any convex function theoretically. Yan and Ouyang (2018) combined wavelet analysis with LSTM to forecast the daily closing price of Shanghai Composite Index. DeepAR method is an industrial probabilistic forecasting algorithm proposed by Amazon in 2019 which is also based recurrent neural network.

Like previous methods, learning system methods have advantages and disadvantages. Advantages: (i) these methods are able to handle intricate patterns because of their complex structure; (ii) Most of these methods do not have preconditions for time series. Disadvantages: (i) huge computation cost due to their self-adapted mechanism and complex structure; (ii) large dataset are required to achieve the satisfied performance; (iii) numbers of hyperparameters need to be tuned for good accuracy.

2.2 Decomposition methods review

Several patterns are displayed by time series data. And it is helpful to decomposing the time series into some components so that patterns can be represented by each component. The decomposition methods can be divided into following two categories, methods used in time series decomposition and signal decomposition.

2.2.1 Time series decomposition methods

The classical decomposition method is very simple method originated from 1920s and is the basis of most methods of time series decomposition. It assumes two forms of decomposition modes, additive decomposition and multiplicate decomposition. Seasonal and Trend decomposition using Loess (STL) is one of derivate methods proposed by Cleveland, Cleveland, McRae, and Terpenning (1990) and is popular in time series decomposition. Time series can be divided into three components, trend component, seasonal component and remainder component when using STL, and the seasonal component is allowed to be changeable and its fluctuation scope can be controlled by users (Hyndman & Athanasopoulos, 2018). Similar methods are X11 decomposition (Dagum, 1975) and SEATS decomposition (Maravall & Gómez, 1994).

2.2.2 Decomposition based on signal decomposition

Time series can be regarded as one kind of signals that can be decomposed into several components like the bellow methods. In the area of signal processing, decomposition methods pay more attention to the frequency of signal series.

Wavelet Transform is most representative technique of them which decomposes signal into several sub-signals with different frequency (Daubechies, 1992). However, the base wavelet to be designated when implements this technique and the popular base wavelets are Haar wavelet (Haar, 1909), Daubechies wavelet (Daubechies, 1992), Biorthogonal wavelet (Mallat, 1999) et al..

From last two decades, Empirical Mode Decomposition (EMD) proposed by Huang et al. (1998) has been widely used in time series decomposition. Similarly, EMD decomposes signal into several components with different frequency and the remainder component, a monotonic series. Compared to Wavelet Transform, EMD is data-driven technique without designating any base function when is implemented. And all the components except the remainder are stationary which meets the need of most statistical forecasting method. But it still has some disadvantages, like mode mixing problem and end effect. To address these disadvantages, Wu and Huang (2009) presented a new Ensemble Empirical Mode Decomposition (EEMD) by adding white noises to each intrinsic mode functions. And this paper adopts EMD as decomposer for financial time series

2.3 Prediction with ensemble learning

For real problems, in order to get better forecasting performance, researcher will combine multiple learning algorithms together instead of using one algorithm alone. Ensemble learning can be regarded as one kind of supervised learning because the general idea of ensemble learning is mapping all prediction results to the real one. According to empirical result, when the learning algorithms are different from each other, the performance of ensemble will be better(Kuncheva & Whitaker, 2003; Sollich & Krogh, 1996). According to the way of generating individual learner, most ensemble learning methods are roughly divided into two categories nowadays: (i) there are strong dependencies between individual learners and learners are generated serially; (ii) no strong dependencies between individual learners and the generation of them can be parallel. The former category is represented by Boosting algorithm and the latter are Bagging and Random Forest.

2.3.1 Boosting

Boosting method is the algorithms which boost the weak learners into a strong learner. The mechanism of these methods is to train a base learner from initial dataset and adjust the distribution of training data so that giving more attention to previous base learners, than train the next base learner based on adjusted training data, and repeatedly the above steps until all base trainers train well.

The most representative algorithm of Boosting methods is Adaptive Boosting algorithm (Freung & Shapire, 1997). This algorithm gives more attention to underfitted samples trained by previous base learners to correct its predecessor. Therefore, the latter base learners will train the harder cases.

Another popular Boosting method is Gradient Boosting. The representative algorithm is Gradient Boosting Decision Tree (GBDT) proposed by Friedman (2001) on the basis of Adaptive Boosting. The difference between two methods is the way of treating the underfitted values of its predecessor. GBDT trains the new base learner to fit the residual errors produced by the previous learner compared to Adaptive Boosting, which adjust the sample weights at each iteration. XGBoost proposed by Chen and Guestrin (2016) is an advanced implementation of Gradient Boosting. This method is able to reach high predictive accuracy and its running time is ten times less than any other gradient boosting algorithm when its paper published. Light Gradient Boosting was proposed by Ke et al. (2017) from Microsoft and was significantly outperformed XGBoost in terms of computational speed and memory occupation. But this article uses XGBoost as base learner instead of ensemble learner.

2.3.2 Bagging and Random Forest

Bagging algorithm proposed by Breiman (1996a) is most representative parallel ensemble learning method which based on Bootstrap sampling, a method of sampling training data from the whole dataset (Tibshirani & Efron, 1993). This method generates sub datasets to train base learners by bootstrap sampling and get the final prediction by average the sub predictions produced by base learners. Because the training of each base learners is individual so that it can be parallel computing to speed up the ensemble learning process.

Random Forest (RF) is an ensemble learning method which also is proposed by Breiman (2001) and is extended from Bagging. RF uses decision tree as base learner and import random selection for features when trains decision tree.

With increasing number of base learners, RF will converge at the level of lower generalization error compared to Bagging. And the training efficiency of RF is superior to Bagging's because all the features need to be considered when Bagging's base learners are constructed.

2.3.3 Stacking

When we have a large-scale training data, a more powerful ensemble learning method is to choose a learner to learn how to combine all results of base learners into one result. Stacking is the representative method of "learning method" (Breiman, 1996b; Wolpert, 1992). The learner of learning combination called meta-learner. The technical details will be introduced in next chapter.

Stacking method is suitable for situation that all base learners are different because the meta-learner is independent of base learners. In this paper, we use stacking method twice. One is stacking all sub predictions of each component after decomposition into one prediction for each component. Another is for all predictions of components into the final prediction of whole financial time series.

3. THEORETICAL PRINCIPLES

3.1 Empirical Mode Decomposition

Empirical Mode Decomposition (EMD) proposed by Huang et al. (1998) is a technique to decompose time series into several intrinsic mode functions, we called them IMFs, and the residual which can be regarded as the trend. The input time series for EMD could be non-stationary series and series with nonlinear relationship and EMD is able to extract intrinsic modes from these series by using the local characteristic of data. One intrinsic mode stands for an IMF, and IMF must be satisfied with two requirements:

(i) the number of extrema should be equal, and the zero-crossing should be no more than one;

(ii) IMF should be symmetrical with zero mean value

The following algorithm will present how EMD decomposes a piece of time series:

(1) given time series $x(t)$, extract all the local maxima and minima, set i as the round number

(2) use minima to generate the lower envelope $e_{min}(t)$ by cubic spline interpolation; similarly, get upper envelope $e_{max}(t)$ with maxima

(3) generate the mean series $m(t)$ by taking the mean of each point from $e_{max}(t)$ and $e_{min}(t)$:

$$m(t) = \frac{e_{max}(t) + e_{min}(t)}{2}$$

(4) take the difference from $x(t)$ and $m(t)$ and get the subtraction $s(t)$:

$$s(t) = x(t) - m(t)$$

(5) if $s(t)$ satisfies the conditions of IMF, consider $s(t)$ as ith IMF and do the following replacement:

$$x(t) = x(t) - s(t)$$

collect $s(t)$ as $c_i(t)$ to indicate the order of IMFs and i plus 1

If not, replace $x(t)$ with $s(t)$

(6) Iterate step (1) to (5) until the residual meet the conditions of some terminal conditions.

According to Huang et al. (2003) 's proposition, one terminal condition should be as follows:

(i) either IMFs $s_i(t)$ or the residual $r(t)$ is so small that the predefined value of significant consequence is more than that;

(ii) or the residual $r(t)$ is monotonic function and no more IMFs can be extracted

In EMD's algorithm, there is limitation for the number of IMFs. Suppose N as the length of $x(t)$, the total number of IMFs should not be more than $\log_2 N$. After getting IMFs $s_i(t)$ and the final residual $r(t)$, the original time series could be presented as follow:

$$x(t) = \sum_{i=1}^{K} s_i(t) + r(t)$$

Where K is the number of IMFs.

In each round of extracting IMFs, all local maxima and minima are picked to form envelopes in time order so that higher frequency appeared in the older IMFs. Therefore, the process of extracting IMFs is a fine-to-coarse procedure that separate frequencies in decreasing order from original time series. Fig 1 shows an example of decompose the stock price series of Apple Inc and well demonstrates this process.

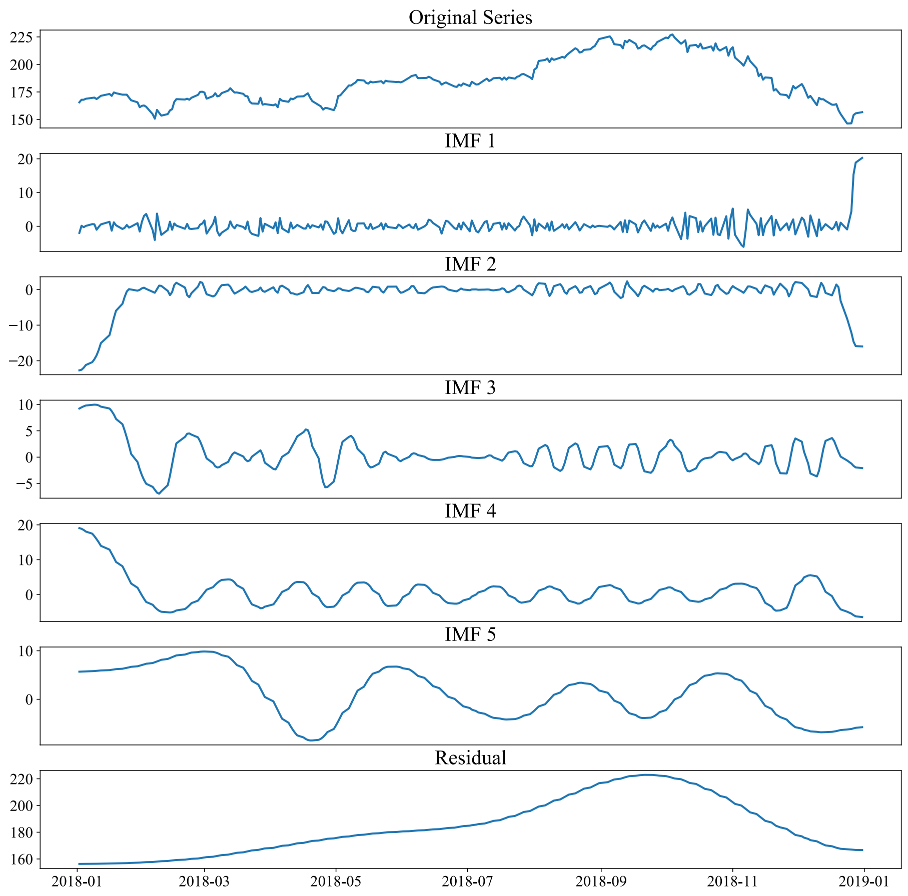

Fig 1 Example of extracted IMFs from time series. The original time series is the daily open price of Apple Inc (AAPL) with the time window between January 1st, 2018 and January 1st, 2019. There are 251 time points in this time series. After processing by EMD algorithm, five IMF components and the residual were obtained, and we could indentify that the frequency or the fluctuation was decreased from IMF1 to IMF5. In the meantime, the residual was rather smoothness and revealed the trend of original series.

3.2 Prediction techniques

3.2.1 LSTM

LSTM is proposed by Hochreiter and Schmidhuber (1997) and is a variation of recurrent neural network (RNN). Therefore, before we introduce LSTM, we should learn about the architecture of RNN.

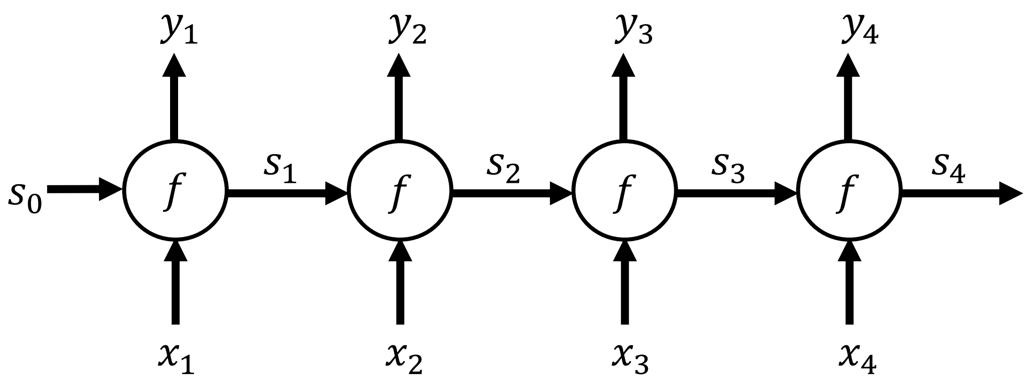

Fig 2 The architecture of recurrent neural network in sequence order

Suppose that x_i is the ith features in dataset, y_i is the ith target value in dataset, s_i is the ith state value, f is a function which maps x_i and s_{i-1} to y_i. As Fig 2 shown, in the initial stage, input the initial state s_0 and feature x_1 than obtain the first output y_1 and the current state s_1. After that, the states s_i will be updated for each new inputting feature while getting new outputs y_i. The relationship between s, x, y and f follows the bellows:

$$y_i, s_i = f(x_i, s_{i-1})$$

In practices, the initial state s_0 would be set to 0 in many cases.

f is composed by two components, f_s and f_y. f_s is the function that maps last period state s_{i-1} and current period input x_i to the current period state s_i. And f_y is the function that maps current state s_i to current output y_i:

$$s_i = f_s(x_i, s_{i-1})$$

$$y_i = f_y(s_i)$$

In the simplest RNN's architecture, f_s and f_y can be wrote like below:

$$s_i = f_s(x_i, s_{i-1}) = \sigma(W_{xs}x_i + W_{ss}s_{i-1})$$

$$y_i = f_y(s_i) = \sigma(W_{ys}s_i)$$

Where σ is sigmoid function and W_{xs}, W_{ss} and W_{ys} is the weight matrix of current input x_i, last period state s_{i-1} and current state s_i respectively.

RNN is able to capture the historical information when input is time series due to its architecture. However, several existence problems appear when RNN is trained. Because neural network is trained by forward propagation and error back propagation, vanishing gradient or gradient explosion will be encountered when doing error back propagation and reduce some historical information in the same time.

For these problems, Long Short-Term Memory (LSTM) neural network proposed the forget gate, input gate and output gate to control the cell state which would help to increase the long memory and deal with short memory. Fig 3 demonstrates this mechanism.

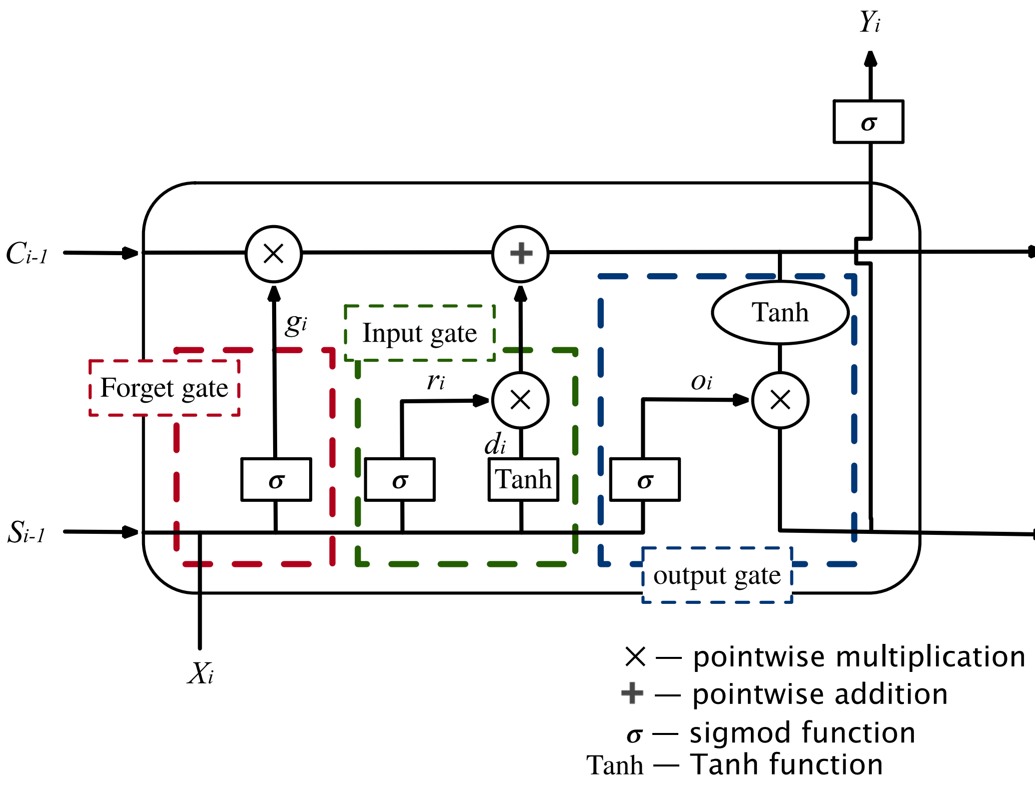

Fig 3 The architecture of LSTM cell

As we can see from **Error! Reference source not found.**, firstly, the forget gate d ecides what information should be forgot for the cell. It accepts s_{t-1} and x_t to produce a vector g_i with values between 0 and 1, which decide how much information of c_{i-1} should discard. 0 stands for forgetting and 1 stands for remembering.

$$g_i = \sigma(W_g[s_{i-1}, x_t])$$

Secondly, the input gate decides what information should be added to the cell. It accepts s_{i-1} and x_i to produce two vectors r_i and d_i. The information stored in r_i and d_i may have a chance to be updated into the information of cell.

$$r_i = \sigma(W_r[s_{i-1}, x_t])$$

$$d_i = tanh(W_d[s_{i-1}, x_t])$$

Then, having g_i, r_i and d_i, the cell could update the old cell information c_{i-1} to the new one c_i. The update rule is that forgetting part of information by pointwise multiplying g_i from the forget gate and adding new information r_i and d_i from the input gate.

$$c_i = g_i * c_{i-1} + r_i * d_i$$

After updating the cell information, produce s_i and the output y_i according c_i and o_i through the output gate.

$$o_i = \sigma(W_o[s_{i-1}, x_t])$$

$$s_i = o_i * \tanh(c_i)$$

$$y_i = \sigma(W_y c_i)$$

3.2.2 SVR

Given the training dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}, y_i \in \mathbb{R}$, we want to learn the model like $f(x) = w^T x + b$ so that $f(x)$ could approach to y as close as possible, w and b are the parameters to be determined.

For sample (x, y), traditional regression model is used to calculate the error based on the difference between model output $f(x)$ and the true value y. The error is 0 if and only if $f(x)$ is exactly equal to y. However, Support Vector Regression (SVR) supposes that we could tolerate there are at most ϵ bias between $f(x)$ and y, i.e., error is calculated only if the absolute value of the difference between $f(x)$ and y. As Fig 4 shown, points fell on the interval-zone with 2ϵ width and center of $f(x)$ are predicted correctly.

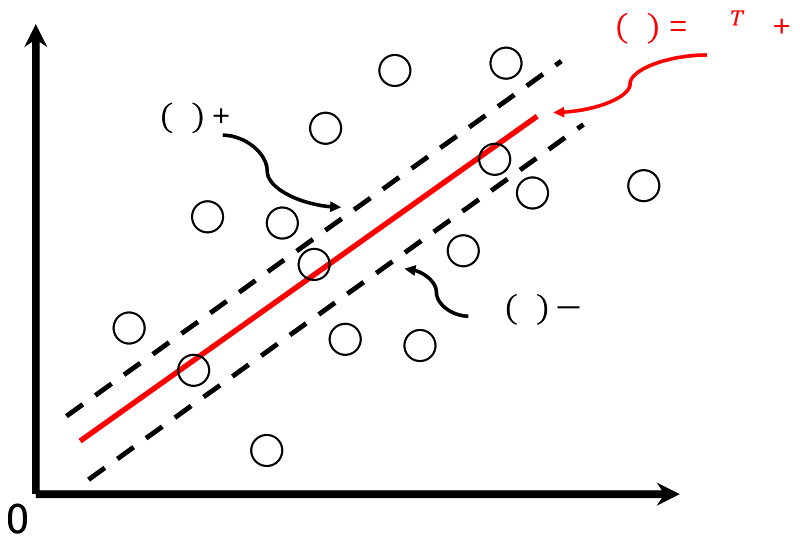

Fig 4 The sketch map of SVR. The red area is the ϵ interval-zone and the error of points which fall in this area are not calculated

Then, SVR problem can be formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \ell_\epsilon(f(x_i) - y_i)$$

Where C is the regularization constant, ℓ_ϵ is ϵ-insensitive loss function as Fig 5 shown.

$$\ell_\epsilon(x) = \begin{cases} 0, & if\ |z| \leq \epsilon; \\ |z| - \epsilon, & otherwise \end{cases}$$

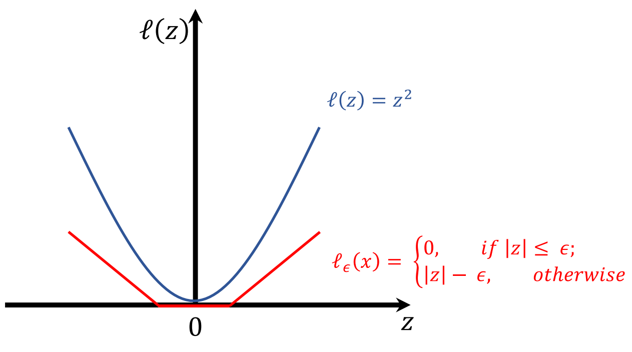

Fig 5 ϵ-insensitive loss function

When import the slack variables ξ_i and $\hat{\xi}_i$, we get the following formula:

$$\min_{w,b,\xi_i,\hat{\xi}_i,} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \hat{\xi}_i)$$

$$s.t.\ f(x_i) - y_i \le \epsilon + \xi_i\ ,$$

$$y_i - f(x_i) \le \epsilon + \hat{\xi}_i\ ,$$

$$\xi_i \ge 0, \hat{\xi}_i \ge 0, i = 1, 2, \dots, m.$$

By importing Lagrange multipliers $\mu_i \ge 0$, $\widehat{\mu_\iota} \ge 0$, $\alpha_i \ge 0$ and $\widehat{\alpha_\iota} \ge 0$, we could get Lagrange function $L(w, b, \alpha_i, \widehat{\alpha_\iota}, \xi_i, \hat{\xi}_i, \mu_i, \widehat{\mu_\iota})$ by Lagrange multiplier method:

$$L\big(w, b, \alpha_i, \widehat{\alpha_\iota}, \xi_i, \hat{\xi}_i, \mu_i, \widehat{\mu_\iota}\big)$$

$$= \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \hat{\xi}_i) - \sum_{i=1}^{m}\mu_i\xi_i - \sum_{i=1}^{m}\widehat{\mu_\iota}\widehat{\xi_\iota}$$

$$+ \sum_{i=1}^{m}\alpha_i(f(x_i) - y_i - \epsilon - \xi_i) + \sum_{i=1}^{m}\widehat{\alpha_\iota},(y_i - f(x_i) - \epsilon - \hat{\xi}_i)$$

By substituting the traditional model and differentiating $L(w, b, \alpha_i, \widehat{\alpha_\iota}, \xi_i, \hat{\xi}_i, \mu_i, \widehat{\mu_\iota})$ with respect to w, b, ξ_i and $\hat{\xi}_i$ as 0, we could get the follow formulations:

$$w = \sum_{i=1}^{m}(\widehat{\alpha_\iota} - \alpha_i)x_i$$

$$0 = \sum_{i=1}^{m}(\widehat{\alpha_\iota} - \alpha_i)$$

$$C = \alpha_i + \mu_i$$

$$C = \widehat{\alpha_\iota} + \widehat{\mu_\iota}$$

Substitute these formulations, we could get the dual problem of SVR

$$\max_{\alpha_i, \widehat{\alpha_\iota}} \sum_{i=1}^{m} y_i(\widehat{\alpha_\iota} - \alpha_i) - \epsilon(\widehat{\alpha_\iota} + \alpha_i)$$

$$-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\widehat{\alpha_i}-\alpha_i)(\widehat{\alpha_j}-\alpha_j)x_i^T x_j$$

$$s.t.\ \sum_{i=1}^{m}(\widehat{\alpha_i}-\alpha_i)=0,$$

$$0 \le \alpha_i, \widehat{\alpha_i} \le C$$

The below progress should be satisfied Karush-Kuhn-Tucker (KKT) condition, i.e.,

$$\begin{cases} \alpha_i(f(x_i)-y_i-\epsilon-\xi_i)=0 \\ \widehat{\alpha_i}(y_i-f(x_i)-\epsilon-\widehat{\xi_i})=0 \\ \alpha_i\widehat{\alpha_i}=0 \\ \xi_i\widehat{\xi_i}=0 \\ (C-\alpha_i)\xi_i=0 \\ (C-\widehat{\alpha_i})\widehat{\xi_i}=0 \end{cases}$$

If and only if $f(x_i)-y_i-\epsilon-\xi_i=0$, α_i can be nonzero value. And similarly, if and only if $y_i-f(x_i)-\epsilon-\widehat{\xi_i}=0$, $\widehat{\alpha_i}$ can be nonzero value. In other words, only if sample (x_i, y_i) do not fall on the ϵ-interval zone, α_i and $\widehat{\alpha_i}$ can be nonzero value. Furthermore, $f(x_i)-y_i-\epsilon-\xi_i=0$ and $y_i-f(x_i)-\epsilon-\widehat{\xi_i}=0$ cannot be satisfied in the same time, thus, there is at least one zero value between α_i and $\widehat{\alpha_i}$.

Then, we get the solution of SVR after back substitutions:

$$f(x)=\sum_{i=1}^{m}(\widehat{\alpha_i}-\alpha_i)x_i^T x+b$$

Samples which make $(\widehat{\alpha_i}-\alpha_i)\neq 0$ is the supported vectors of SVR and is fell out of ϵ-interval zone. Obviously, the supported vectors of SVR are part of training data and its solutions is still sparsity.

According to KKT condition, for every sample (x_i, y_i), it satisfy $(C - \alpha_i)\xi_i = 0$ and $\alpha_i(f(x_i) - y_i - \epsilon - \xi_i) = 0$. Then, after getting α_i, if $0 < \alpha_i < C$, it must exist $\xi_i = 0$, and,

$$b = y_i + \epsilon - \sum_{j=1}^{m} (\widehat{\alpha_i} - \alpha_i) x_j^T x_i$$

Thus, after getting α_i, we could get b by selecting any sample which satisfy $0 < \alpha_i < C$. In practice, it usually adopts a more robust method, that is, picking multiple or all samples satisfied $0 < \alpha_i < C$ to get b and then take an average.

If we consider the feature mapping form, and accordingly, the form of SVR is:

$$f(x) = \sum_{i=1}^{m} (\widehat{\alpha_i} - \alpha_i) \kappa(x, x_i) + b$$

Where $\kappa(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel function.

3.2.3 XGBoost

XGBoost which stands for "Extreme Gradient Boosting" proposed by Chen and Guestrin (2016), is one of Boosting algorithms. The general idea of XGBoost is to combine weak learners to form strong learner. Because XGBoost is a boosting tree method which integrates lots of tree model to build a strong learner and its base tree model is Classification and Regression Tree (CART) model, we would introduce CART before knowing the principle of XGBoost.

Classification and Regression Tree

CART is a binary tree which splits features and as its children nodes. For example, the current tree node is splitting based on j-th feature with supposition that the feature value which is less than s is divided to the left subtree and the one greater than s is on the right subtree:

$$R_1(j,s) = \{\, x \mid x^{(j)} \leq s \,\}$$

$$R_2(j,s) = \{\, x \mid x^{(j)} > s \,\}$$

Essentially, CART divides its samples on feature dimension. But the optimizing of this division is a NP-hard problem, therefore, heuristic algorithm is decision model. A classical objective function of CART is:

$$\sum_{x_i \in R_m} \left(y_i - f(x_i)\right)^2$$

Thus, to find the best divided feature j and divided point s, we solve this transformed objective function:

$$\min_{j,s}[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2]$$

By traversing all the divided points of all features, we can find the best divided feature and divided point, then, get the regression tree.

The idea of XGBoost algorithm

The idea of XGBoost is continually adding trees and continually splitting features to grow trees. Each new added tree is to learn a new function to fit the residual of last forecasting and we get k trees after training. When we forecast one sample, each feature will get a score from the leaf node of one tree and the forecast value is the sum of scores of all trees.

$$\hat{y} = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)$$

$$where\ F = \{ f(x) = \omega_{q(x)} \} \ (q: R^m \to T, \omega \in R^T)$$

$\omega_{q(x)}$ is the score of leaf node, $f(x)$ is one of regression trees.

The objective function of XGBoost can be defined like bellows:

$$Obj = \sum_{i=1}^{n} l(y_i + \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|\omega\|^2$$

The objective function is composed by two parts, l and Ω. l is the loss function that evaluates the difference between predicted score and real score. Ω is the regularization part that contains two parts: T stands for the number of leaf nodes and ω stands for the score of leaf nodes. γ can control the number of leaf nodes and λ can prevent the score from overwhelming so that it would not be overfitting.

As mentioned above, the newly grown tree is fitting the residual of last forecasting, that is, after t trees grown, the score can be:

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

At the same time, the objective function can be:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Obviously, we need to find an objective function which allows f_t to be minimized. What XGBoost do is to do Tayler second order expansion where $f_t = 0$. Therefore, the objective function is approximate to:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [\, l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)] + \Omega(f_t)$$

And g_i is the first-order derivative of l, h_i is the second-order derivative.

$$g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$$

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$$

Because the residual between the forecast scores of the previous $t-1$ trees and y does not affect the optimizing for objective function, they can be removed directly. The simplified objective function is:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [\, g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)] + \Omega(f_t)$$

The above formula is to add up all the losses. Because every sample will reach one leaf node, we can regroup samples of reaching the same leaf node. The process is:

$$obj^{(t)} \simeq \sum_{i=1}^{n} \left[\, g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)\right] + \Omega(f_t)$$

$$= \sum_{i=1}^{n} \left[g_i \omega_q(x_i) + \frac{1}{2} h_i \omega_{q(x_i)}^2\right] + \gamma T + \frac{1}{2}\,\lambda \sum_{j=1}^{T} \omega_j^2$$

$$= \sum_{j=1}^{T} \left[\left(\sum\nolimits_{i \in I_j} g_i\right) \omega_j + \frac{1}{2}\left(\sum\nolimits_{i \in I_j} h_i + \lambda\right) \omega_j^2\right] + \gamma T$$

After rewriting the formula above, we could rewrite the objective function to a quadratic equation about the score of leaf node ω so that it is easy to solve the best ω and objective function value by using vertex formula. Thus, the best ω and objective function is:

$$\omega_j^* = -\frac{G_j}{H_j + \lambda}$$

$$obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

In the derivation above, we know how to calculate the score of each leaf node after a regression tree. However, we still not know how to build a regression tree, more specifically, how to find the best feature and the best splitting point. As mentioned above, it is a NP-hard problem that build a decision tree based on feature space, i.e., it is impossible that traversing all the tree nodes. Thus, XGBoost adopts greedy algorithm which is used in CART to traverse all features and all splitting points and uses the objective function above as evaluation function. Specific approach is to set a threshold. Only if the gain of splitting is greater than the threshold, the tree is split, which prevents tree from getting deeper. In the meantime, we could set the max depth of tree and the stop criterion.

XGBoost also proposed two ways to prevent overfitting: Shrinkage and Column Subsampling. Shrinkage is the method that multiplying a shrink weight η with the score of each leaf node in each iteration so that reducing the importance of each tree and leaving optimize zoom to later trees. Column Subsampling is similar to the way to select partial features in Random Forest. There are two approaches in Column Subsampling. One is random subsampling by layer that select partial features before splitting, then traverse these features to settle the best splitting point. Another is random selection for features to traverse for splitting. In practice, the former's performance is better.

Advantages of XGBoost

XGBoost is a very popular model that widely used in data science competition and industry because of its advantages:

(1) Multiple strategies are used to prevent overfitting, such as regularization, Column Subsampling and Shrinkage;

(2) The objective function is optimized by the second derivative of loss function;

(3) It supports parallel processing which is the flash point of XGBoost;

(4) It can deal with sparse data.

3.2.4 Prophet

In business scenario, forecasting is very necessary demand for most companies. For examples, the number of commodity orders need to be predicted to control the inventory so that saves the storage and reduces depreciation. However, forecasting is a tough nut for some enterprises to crack due to the requirement of deep domain knowledge and statistic knowledge about time series models and methods for analyst. Based on this background, Taylor and Letham (2018) from Facebook proposed Prophet algorithms, the algorithm defaults the workflows of modeling time series and the parameters, to help non-experts to build feasible model according their demands.

Many business time series exist some common temporal features: multiple strong seasonality, trend changes, outliers and holiday effects. Prophet proposed an additive model like below:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

Where $g(t)$ represents the trend function, $s(t)$ is periodic changes function, $h(t)$ represents the effects of holidays and the error term ε_t.

The Trend Model

There are two important trend techniques in Prophet. One is saturating growth model, another is piecewise line model

Saturating growth model is designed to deal with nonlinear trend. The basis form of this model is:

$$g(t) = \frac{C}{1 + \exp\left(-k(t-m)\right)}$$

Where C is the carrying capacity, k is the growth rate and the offset parameter m

In practice, C, k and m are not constants but the values changed by time. Therefore, Prophet supposed them as time functions

Prophet supposed there are S changepoints in corresponding timestamp s_j $(1 \leq j \leq S)$. Then, add the change rate in each changepoint. Suppose we have a vector $\delta \in \mathbb{R}^S$ and δ_j represents the change rate at timestamp s_j. Then, we get the growth rate $k + \sum_{j:t>s_j} \delta_j$ at timestamp t. With using sign function $a(t) \in \{0,1\}^S$, that is,

$$a_j(t) = \begin{cases} 0, if\ t \geq s_j \\ 0, otherwise \end{cases}$$

Then, we could gain growth rate $k + a(t)^T\delta$

For offset parameter m, we need to adjust the endpoints of the segments and get

$$\gamma_j = (s_j - m - \sum_{l<j} \gamma_l)(1 - \frac{k + \sum_{l<j} \delta_l}{k + \sum_{l\leq j} \delta_l})$$

The carrying capacity C need to be set by user when the growth parameter in Prophet is set to “growth”. Therefore, we reset these parameters as below:

$$C := C(t)$$

$$k := k + a(t)^T\delta$$

$$m := m + a(t)^T\gamma$$

And the mathematical form of saturating growth model is:

$$g(t) = \frac{C(t)}{1 + \exp(-(k + a(t)^T\delta) \cdot \left(t - (m + a(t)^T\gamma)\right))}$$

Where $a(t) = \left(a_1(t), \dots, a_S(t)\right)^T$, $\delta = (\delta_1, \dots, \delta_S)^T$ and $\gamma = (\gamma_1, \dots, \gamma_S)^T$.

Piecewise line model is designed to deal with linear trend. The basis form of this model is:

$$g(t) = kt + m$$

Likewise, k is the growth rate and m is the offset term.

We already got the functions of k and m hereinbefore, and the basis form can be rewritten as below:

$$g(t) = (k + a(t)^T\delta) \cdot t + (m + a(t)^T\gamma)$$

However, γ here is different than that of saturating growth model.

$$\gamma = (\gamma_1, \dots, \gamma_S)^T, \gamma_j = -s_j\delta_j$$

Seasonality

Prophet approximates the seasonality with a standard Fourier series:

$$s(t) = \sum_{n=1}^{N} (a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right))$$

In Prophet's paper, it is suitable that $N = 10$ and $P = 365.25$ for series with yearly periodicity; For series with weekly periodicity, authors recommend $N = 3$ and $P = 7$. The parameters above can form the column vector:

$$\beta = (a_1, b_1, \dots, a_N, b_N)^T$$

When $N = 10$,

$$X(t) = [\cos\left(\frac{2\pi(1)t}{365.25}\right), \ldots, \sin\left(\frac{2\pi(10)}{365.25}\right)]$$

Thus, the season component is

$$s(t) = X(t)\beta$$

The initialization of β follows $\beta \sim Normal(0, \sigma^2)$. σ is controlled by the variance "seasonal_prior_scale" in Prophet. With larger value of σ, the seasonal effect is more significant and vice versa.

Holidays and Events

In realistic, there are lots of holidays and events which have influences on people's behavior and holidays in different country can be various. Because the effect of each holiday is different, each holiday can be regarded as an independent model with different window length. For i-th holiday, D_i represents the period of that holiday. To display the effect of holiday, we need an indicator function and a parameter k_i. Supposed we have L holidays, then

$$h(t) = Z(t)K = \sum_{i=1}^{L} k_i \cdot 1_{\{t \in D_i\}}$$

where $Z(t) = (1_{\{t \in D_1\}}, \ldots, 1_{\{t \in D_L\}})$, $K = (k_1, \ldots, k_L)^T$ and $K \sim Normal(0, v^2)$

v is set by the variance, "holidays_prior_scale", with default value 10. With larger value of v, more holiday effect on the model and vice versa.

Model Fitting

For the interpretation above, we got the model with trend, seasonality and holiday item:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon$$

In Prophet, it used the L-BFGS function from pyStan, a python library for Bayesian inference, to fit the model above.

3.2.5 DeepAR

DeepAR proposed by Salinas et al. (2019) is a time series forecasting method based on recurrent neural network. It has embedded into Amazon SageMaker, the machine learning cloud platform of AWS, and GluonTS, the opensource time series forecasting library of Amazon.

Traditional time series forecasting techniques such as ARIMA, Holt-Winters' are focus on modeling one-dimensional time series without additional features. Furthermore, the objective of traditional time series forecasting methods is the values of predictive period. DeepAR, by contrast, predict the probabilistic distribution of each predictive step which is more meaningful in some specific scenarios such as sales forecasting of products in retail.

Model of DeepAR

Suppose $z_{i,t}$ to represent the value of i-th time series in timestep t, and $x_{i,t}$ to represent the corresponding feature. t_0 is the current time. The distribution of $z_{i,t}$ of DeepAR, an algorithm based on autoregressive recurrent neural network, is modeled by likelihood function $\ell(z_{i,t}|\theta_{i,t})$.

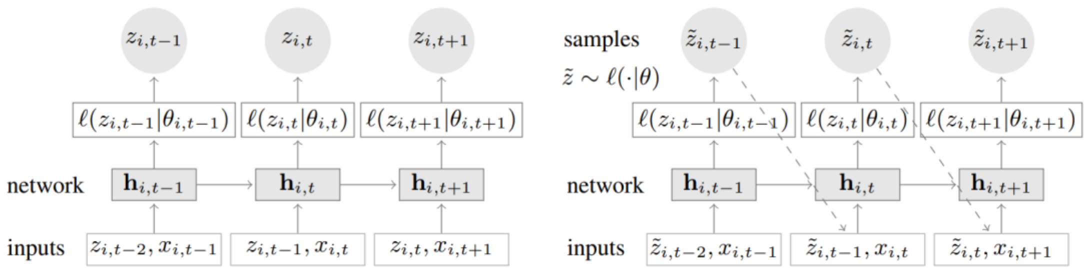

Fig 6 The training and prediction prosedures of DeepAR. The graph on the right is the architechture of training network. The left is that of forecasting.

As Fig 6 shown, in the training procedure, for each timestep t, feature $x_{i,t}$, the last observation $z_{i,t-1}$ and the state of last timestep $h_{i,t-1}$ are the input of model. Firstly, compute the current state $h_{i,t}$ and parameter $\theta_{i,t}$ of likelihood function $\ell(z_{i,t}|\theta_{i,t})$ as below:

$$h_{i,t} = h(h_{i,t-1}, z_{i,t-1}, x_{i,t})$$

$$\theta_{i,t} = \theta(h_{i,t})$$

Then, learn the model parameter by maximizing the log likelihood $\mathcal{L}$:

$$\mathcal{L} = \sum_i \sum_t log\ell(z_{i,t}|\theta(h_{i,t}))$$

After training, it feed the historical data with timestep $t < t_0$ back to the network and get the initial state h_{i,t_0-1}. Then, we get the prediction by ancestral sampling: for t between t_0 and T, we get the random sample $z_{i,t} \sim \ell(\cdot\, | \theta_{i,t})$ at each timestep. This sample will be reused as the input of next timestep. Finally, get the sample values of $t_0 \sim T$ by repeating the above steps. With these samples, we compute the required targets such as quantile and expectation.

The specific form of $\theta(h_{i,t})$ is depended on the likelihood function $\ell(z_{i,t} | \theta_{i,t})$, and the selection of likelihood function is depended on the statistic features of its data. If chosen Gaussian distribution, $\theta = (\mu, \sigma)$, the mean and variance. The specific form of likelihood function shown as below:

$$\ell_G(z|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(z-\mu)^2}{2\sigma^2}\right)$$

$$\mu(h_{i,t}) = w_\mu^T h_{i,t} + b_\mu$$

$$\sigma(h_{i,t}) = \log\left(1 + \exp(w_\sigma^T h_{i,t} + b_\sigma)\right)$$

Where w_μ and w_σ is the output of fully connected networks with input μ and σ respectively.

4. METHODOLOGY

4.1 framework

This paper proposes three-stage forecasting technique which combines time series decomposition and model stacking and the method framework is displayed in Fig. 6

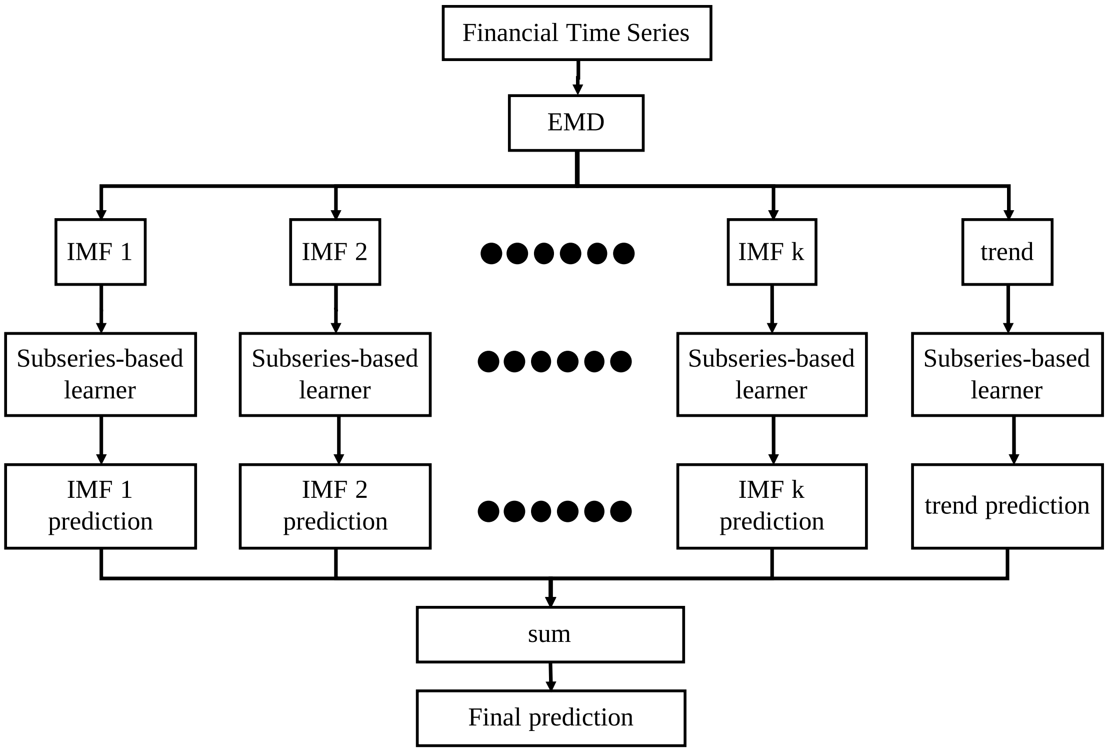

Fig. 1 proposed forecasting framework

The original financial time series will be decomposed into k IMF components and the remainder, the trend, by Empricial Mode Decomposition. Then, these decomposed components will be input into the subseries-based learner which will be mentioned in section 4.2 to produce the prediciton of each decomposed components. Finally, these sub-perdiciton will be sent to train a linear regression model so that model can learn the weight of each component to output the final prediction of original financial time series.

The reason we use linear regression to integrate the sub-preditions is that during the process by EMD, the sum of each components is equal to the value of original series which has mentioned in section 3.1 , that is, the reconstruction process is a linear process so that we could use a linear model to simulate the reconstruction and linear regression model is our chose.

4.2 subseries-based learner

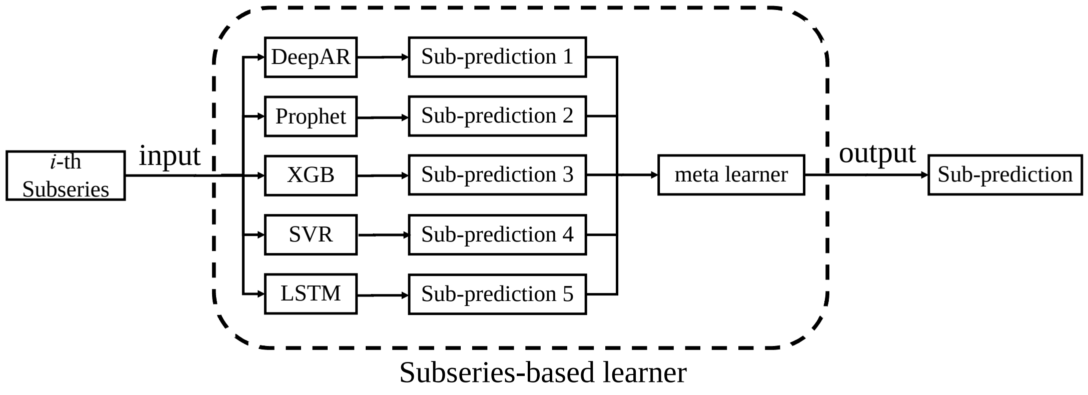

Fig 7 the architecture of subseries-based learner

As Fig 7 shown, the i-th subseries will be input into subseries-based learner. Then, five models (DeepAR, Prophet, XGBoost, SVR and LSTM) will be trained and produce sub-predictions. In the same time, meta learner will be trained which will be illustrated in section 4.3. After training, the sub-predictions will be integrated by trained meta-learner and produce the sub-prediction of i-th subseries.

4.3 Training process of meta learner

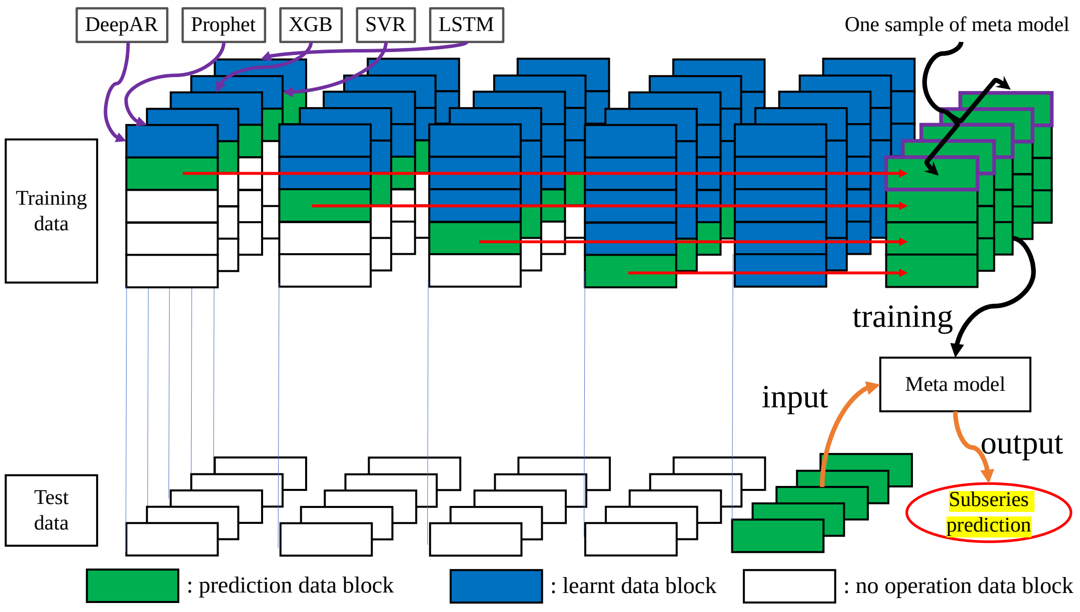

Fig 8 training process of meta leaner

As Fig 8 shown, in training period, we divide training data into N equal-sized data blocks. Then, we take the increasing size of data blocks as training set of each prediction models and get the predictions of each prediction model after iteratively training, which form the training set of meta model, and we call it meta-training data. After that, we use the original training data to train our prediction models and the forecasting result. Finally, we train meta model with meta-training data, and get the meta result by inputting the prediction of test data into the meta model.

For example, supposed we have four prediction models, and five folds are divided in the original training data. At the first round, the first fold is inputted to each prediction model, and we get the predictions of the second fold. In round two, we get the predictions of third fold by inputting the first and second folds. At the third and fourth round, the predictions of the fourth and the fifth folds are obtained by inputting the first to third folds and the first to fourth folds respectively. In above rounds, we get the training data of meta learning and we call it meta-training data. Then, meta-training data will be packed to meta model for training. The features of one sample of meta-training data are the fourth predictions and the label is the true value of that timestep. After training, we use the whole original training data to get the prediction of test data and get the final prediction of test data by inputting the predictions to meta model.

Mathematical Formula

After pre-training of k prediction models, we get meta dataset with N samples.

$$D_{meta} = \{(\{p_{j,i}\}, y_i)\}$$

$$j = 1,2, \dots , k; i = 1,2, \dots , N$$

Essentially, the meta learner model is a weighted average model:

$$f_{wa}(p_1, p_2, \dots , p_k, a_1, a_2, \dots , a_k) = \sum_{i=1}^{k} a_i \, p_i$$

k is the number of prediction models and the constraint of model shows as follow:

$$\sum_{i=1}^{k} a_i = 1, 0 \le a_i \le 1$$

And we get $\{a_1, a_2, \dots , a_k\}$ by minimizing following formula:

$$\min_{a_1,a_2,\dots,a_k} \frac{1}{N} \sum_{N}^{i=1} (y_i - f_{wa})^2$$

4.4 Reconstruction

After getting the final predictions of each IMF and the trend, we add them up:

$$f_{series} = f_{imf_1} + f_{imf_2} + \cdots + f_{imf_m} + f_{trend}$$

where m is the number of IMFs.

The reason of addition is that in the decomposition process, EMD decompose series into several IMFs and a trend and the sum of them is the original value. We suppose the prediction of each IMF and trend are reasonable, then, the sum of them is supposed to be the prediction of original value.

5. EXPERIMENTS

5.1 Datasets

We conduct experiments to verify the performance of our framework on three public financial datasets. The statistic information of these datasets is described in Table 1.

Table 1 The statistic information of the Dataset

Dataset	Length	Granularity	Date Range
Stock Index	8477	daily	1986 – 2018
Exchange-rate	6894	daily	2000 - 2019
Oil Price	8590	daily	1987 - 2020

5.1.1 Stock Index Dataset

This dataset[1] collects the daily closing price of S&P500 index from 1986 to 2018. It starts from January 2nd, 1986 and ends at January 29th, 2018 with total 8477 business days. Fig 9 and Fig 10 shows the plotting of the daily closing price and daily return of this dataset, respectively.

[1] https://www.kaggle.com/pdquant/sp500-daily-19862018

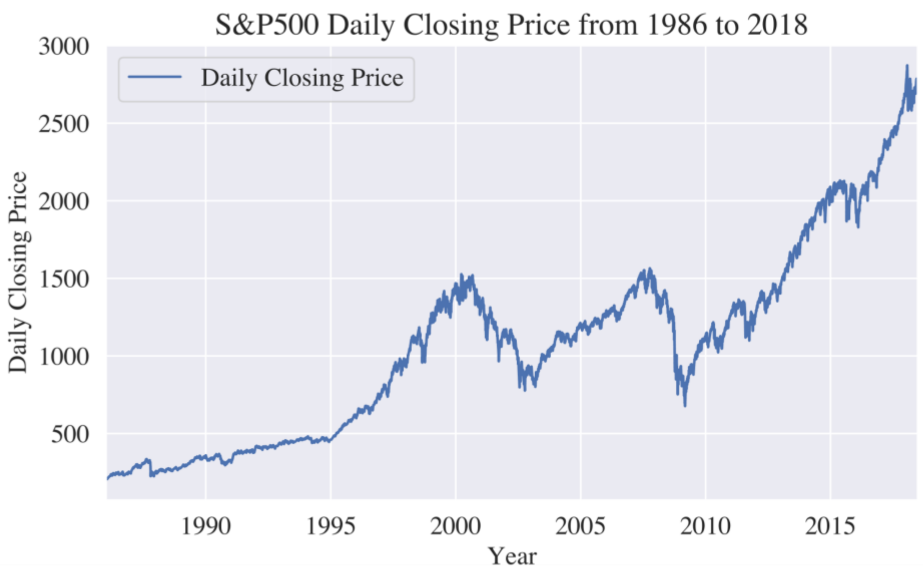

Fig 9 Daily closing price of S&P500 Index from 1986 to 2018

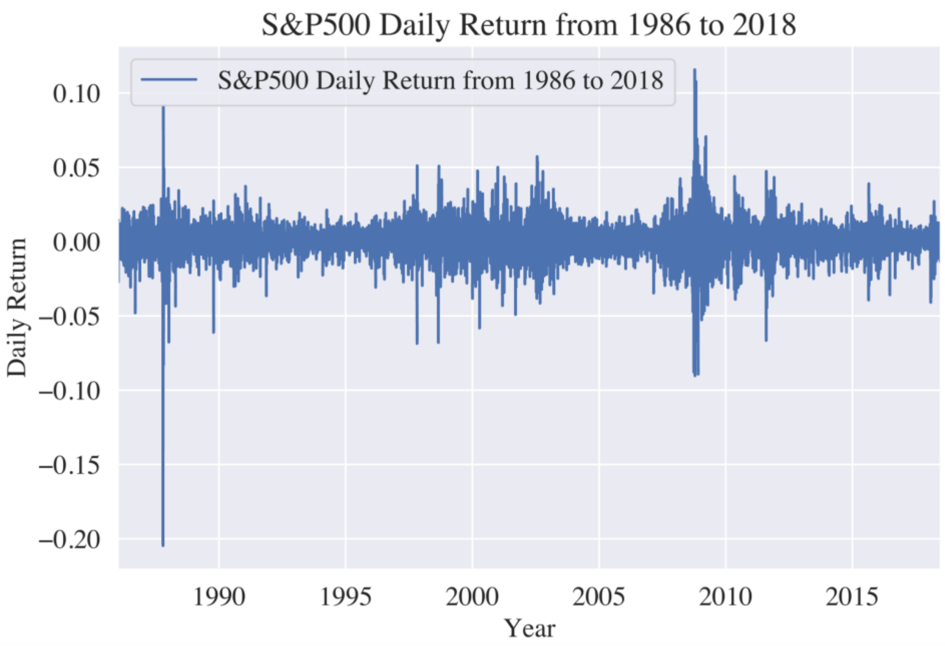

Fig 10 Daily Return of S&P500 Index from 1986 to 2018

5.1.2 Exchange-rate Dataset

Exchange-rate dataset[2] collects hourly prices and technical indicators of GBPUSD exchange rate from 2000 to 2019. It starts from June 4th, 2000 and ends at April 19th, 2019 with 120842 hourly data. We average the hourly data by day and obtains 6894 daily open prices. Fig 11 and Fig 12 shows the plotting of the daily closing price and daily return of this dataset, respectively.

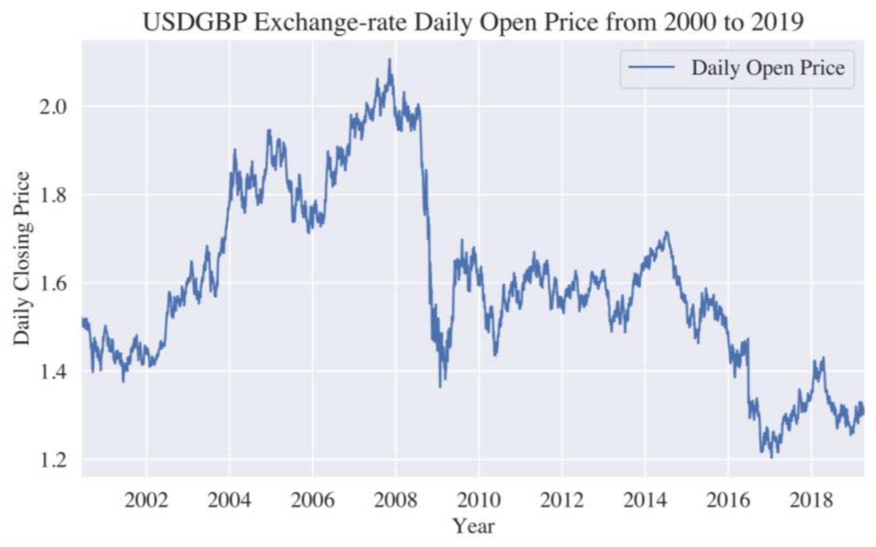

Fig 11 Daily Open Price of USDGBP Exchange-rate from 2000 to 2019

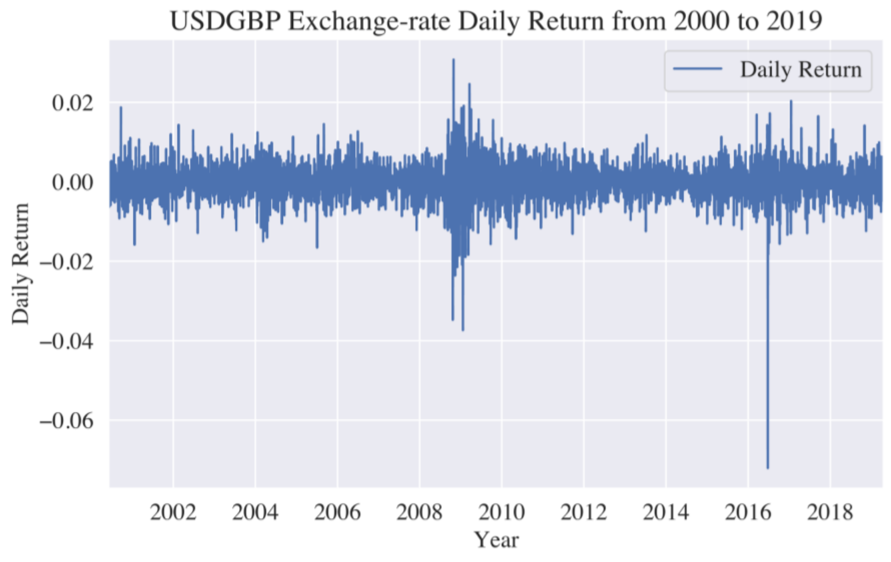

Fig 12 Daily Return of USDGBP Exchange-rate from 2000 to 2019

[2] https://www.kaggle.com/cfchan/hourly-gbpusd-w-technical-indicators-20002019

5.1.3 Oil Price Dataset

This dataset[3] collects daily Brent Oil prices, which was retrieved from the U.S. Energy Information Administration, from May 20th, 1987 until February 25th, 2020 with total 8590 business days. Fig 13 and Fig 14 shows the plotting of the daily price and daily return of this dataset, respectively.

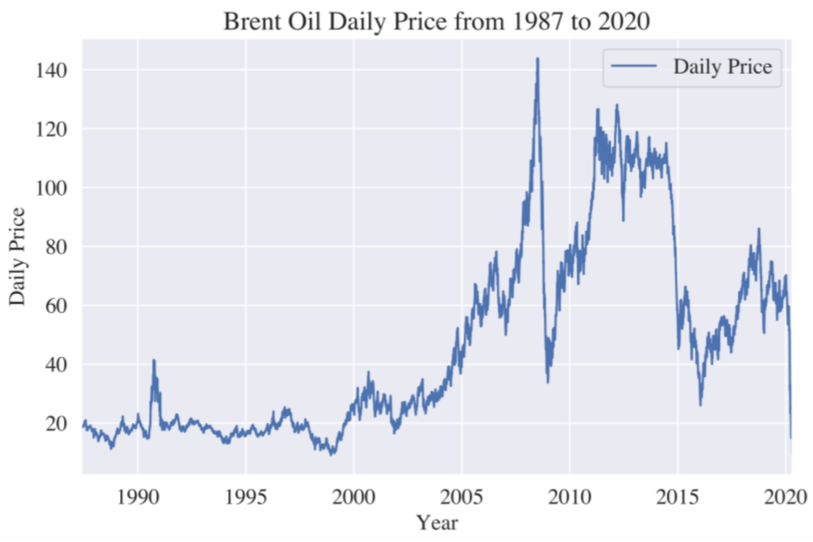

Fig 13 Daily Price of Brent Oil from 1987 to 2020

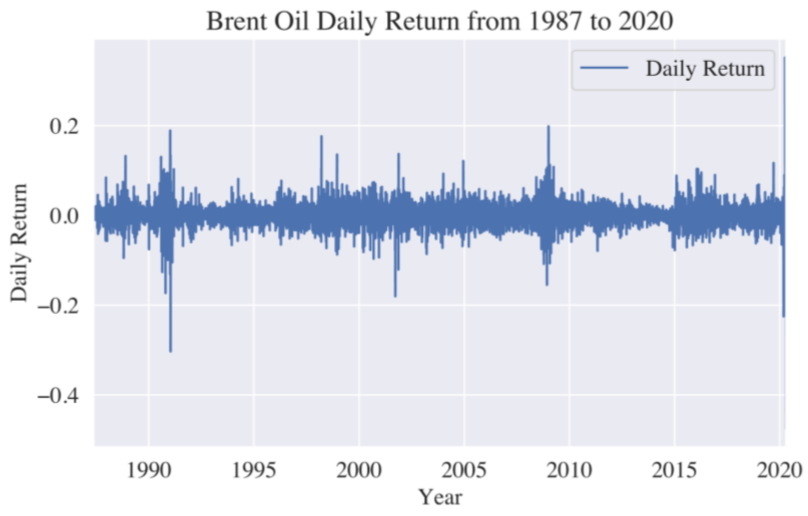

Fig 14 Daily Return of Brent Oil Price from 1987 to 2020

[3] https://www.kaggle.com/mabusalah/brent-oil-prices

5.2 Empirical Mode Decomposition Analysis

We next study the situations of three datasets after decomposing by Empirical Mode Decomposition.

5.2.1 Stock Index Dataset

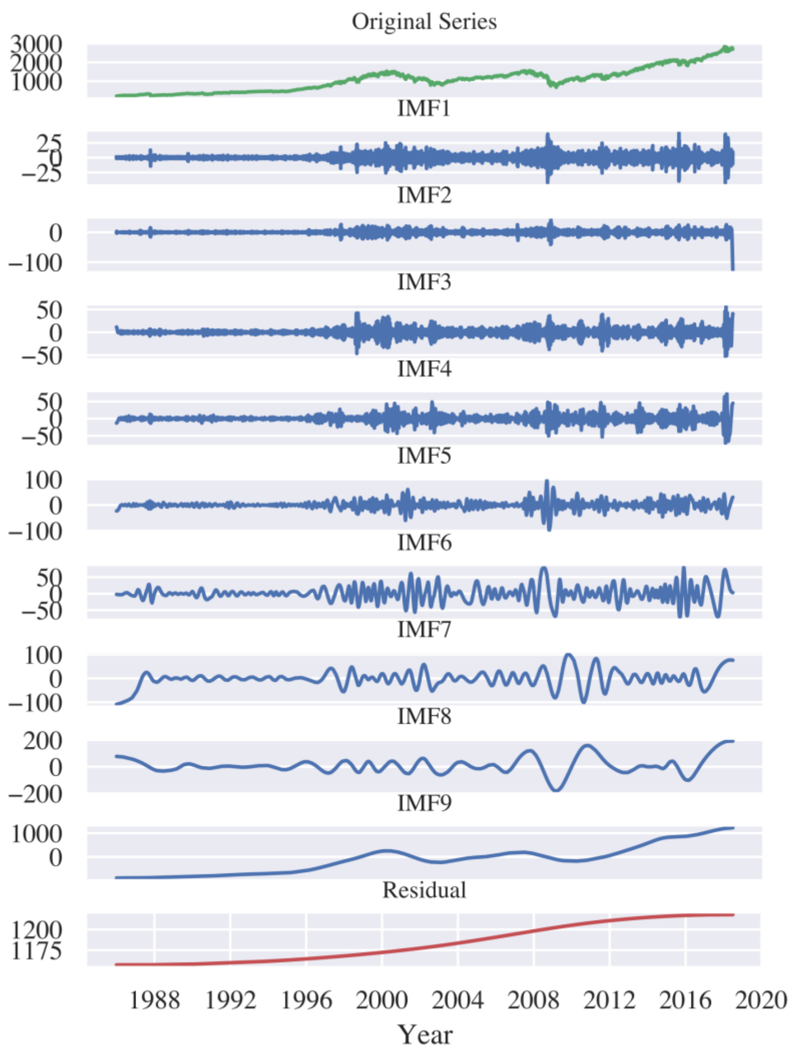

Fig 15 Plotting of Daily Closing Price of S&P500 Index from 1986 to 2018 and its IMFs and residual after EMD decomposition

From Fig 15, we easily know that the residual of this dataset is increasing, which means S&P500 index have an upward trend. There are nine IMFs after EMD decomposition, five of them are highly fluctuant and the frequency of all IMFs are descending from top to bottom.

5.2.2 Exchange-rate Dataset

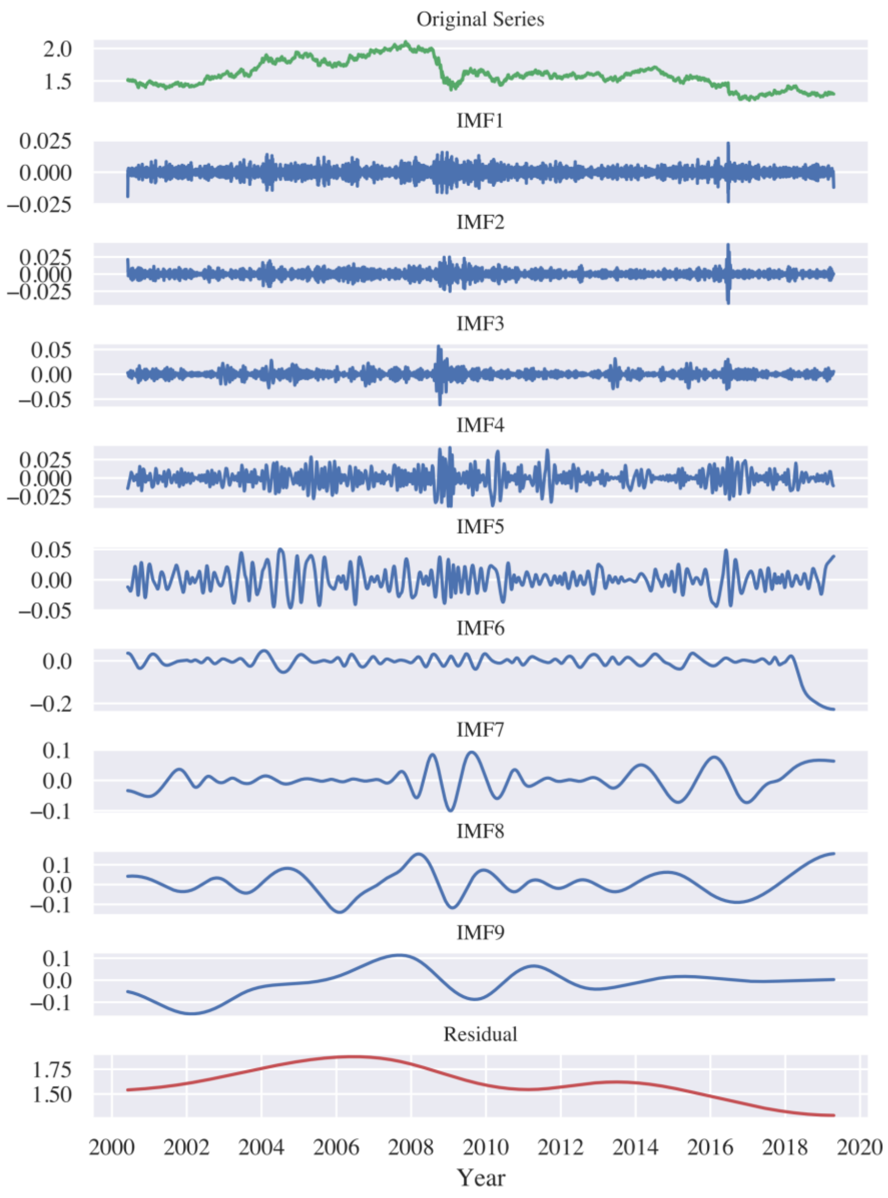

Fig 16 Plotting of Daily Open Price of USDGBP Exchange-rate from 2000 to 2019 and its IMFs and residual after EMD decomposition

Fig 16 shows that there are nine IMFs and one residual after EMD decomposition. We notice that the residual is decreased during this period which means the exchange rate have a downward trend, i.e., US dollar is cheaper compared to the pound. Similarly, we find that the frequency of all IMFs is lower from top to bottom.

5.2.3 Oil Price Dataset

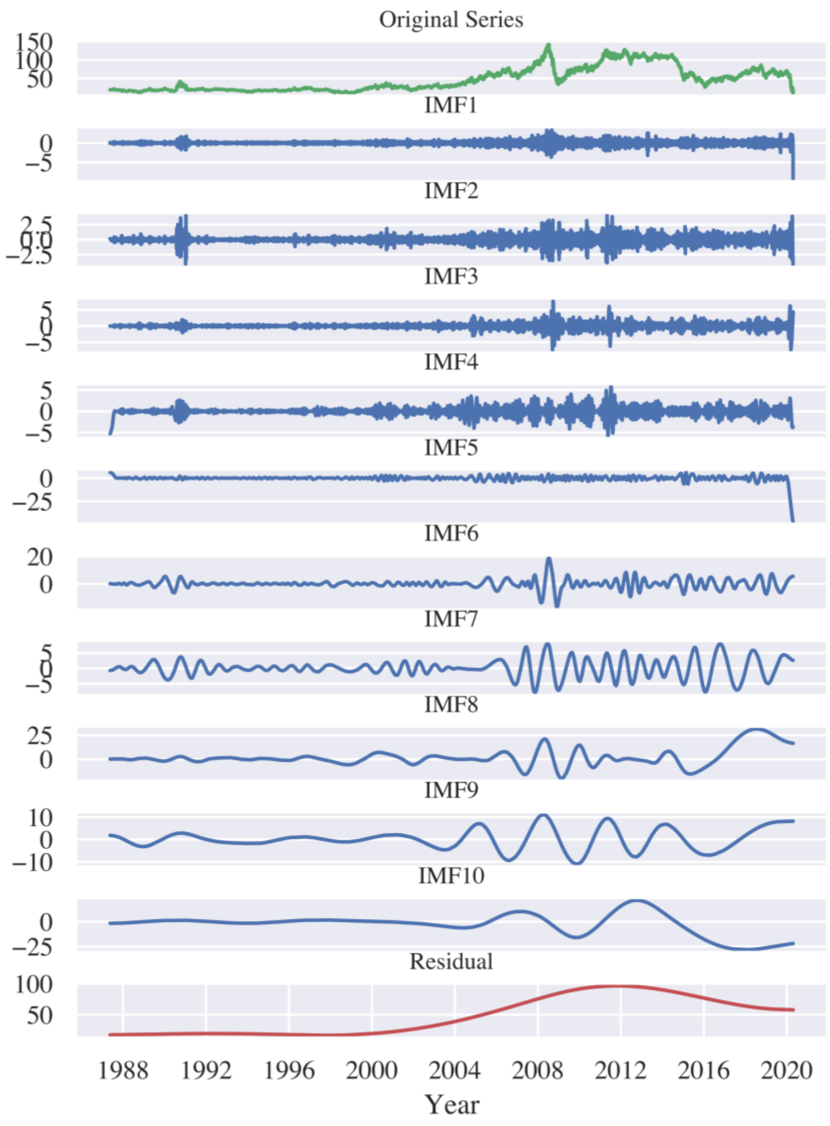

Fig 17 Plotting of Daily Price of Brent Oil Price from 1987 to 2020 and its IMFs and residual after EMD decomposition

Fig 17 shows that there are ten IMFs and one residual after decomposing by EMD algorithm. The residual slowly increased during 1988 to 2004, followed by a rapid ascend from 2004 to 2012, and slightly dropped after 2012, which reflects the long-term trend of Brent Oil price. We also can observe that the decreasing frequency of IMFs from top to bottom.

5.3 Experimental Settings

5.3.1 Experiment environment

All experiments are conducted on a Linux server with 96xIntel Xeon® 2.70GHz CPUs, 512GB RAM. We use *Python* as our program language and all models are implemented with open source libraries and deep learning framework such as *sklearn*, *tensorflow*.

5.3.2 Compared Models

In this paper, we proposed a forecasting framework based on EMD and machine learning models. To evaluate the effect of our framework, here, we compare our models using our framework with those without framework respectively.

5.3.3 Evaluation Metrics

Two metrics are used to evaluate the performance of each models. One is the traditional forecasting metric used in regression problems: mean square error (MSE). Because the MSE metric don't have much practical reference value when conduct financial strategies and trading, we employ the direction accuracy (DA) of return to evaluate the trend accuracy of predictive return. The formulations of MSE and DA are defined as follows:

$$MSE = \frac{\sum_{i=1}^{N} |y_i - \widehat{y_i}|^2}{\bar{y}}$$

$$r_i = \frac{y_i - y_{i-1}}{y_{i-1}}$$

$$d_i = \begin{cases} 1, & r_i * \widehat{r_i} > 0 \\ 0, & r_i * \widehat{r_i} \leq 0 \end{cases}$$

$$DA = \frac{\sum_{i}^{N} d_i}{N}$$

Where y_i, $\widehat{y_i}$ is true price and predictive price respectively, $\bar{y}$ is the average of true value and N is the number of predictions; r_i, $\widehat{r_i}$ is the true return and predictive return respectively, d_i is the direction accuracy of i^{th} prediction.

5.3.4 Parameter Settings

As mentioned before, five machine learning models are employed with the framework to evaluate its performance. For SVR, XGBoost, LSTM models, we generate inputs with sliding window methods with the sliding window size of 60 for each dataset.

Each dataset is split into training sets and validation sets by ration 9:1. The last 30 data points are left for testing. The validation sets are used to tune the hypermeters of models

6. RESULTS & ANALYSIS

6.1 Stock Index Prediction

6.1.1 Forecasting Result

The forecasting results of Stock Index dataset are reported at Table 2, which shows as bellow.

Table 2 Stock Index Prediction results(in MSE and DA) of proposed framework and individual methods before EMD decomposition and after EMD decompositon.

	MSE		DA	
Method	Before EMD	After EMD	Before EMD	After EMD
DeepAR	4.170 x 10^3	7.187 x 10^3	41.4%	44.8%
Prophet	1.107 x 10^4	1.012 x 10^4	48.3%	48.3%
XGB	2.110 x 10^3	5.318 x 10^2	24.1%	58.6%
SVR	2.749 x 10^6	2.431 x 10^6	51.7%	55.2%
LSTM	7.145 x 10^3	4.405 x 10^3	41.4%	34.5%
META	2.358 x 10^3	5.116 x 10^2	24.1%	69.0%

From Table 2, we can easily know our proposed framework (META after EMD) get the lowest MSE and the highest DA and the effect of EMD is significant that the MSE of our framework achieved 511.6 which perform those before EMD 361%, in the meantime, DA increased from 24.1% to 69.0%, nearly a triple improvement. Besides, during the EMD, there are different degrees of improvements among most models (Prophet, XGBoost, SVR and LSTM) in evaluation of MSE and the same situation happens in DA metric among DeepAR, Prophet, XGBoost, SVR models. All these experiment results demonstrate our proposed framework works to improve the forecasting results.

6.1.2 Overall Comparison

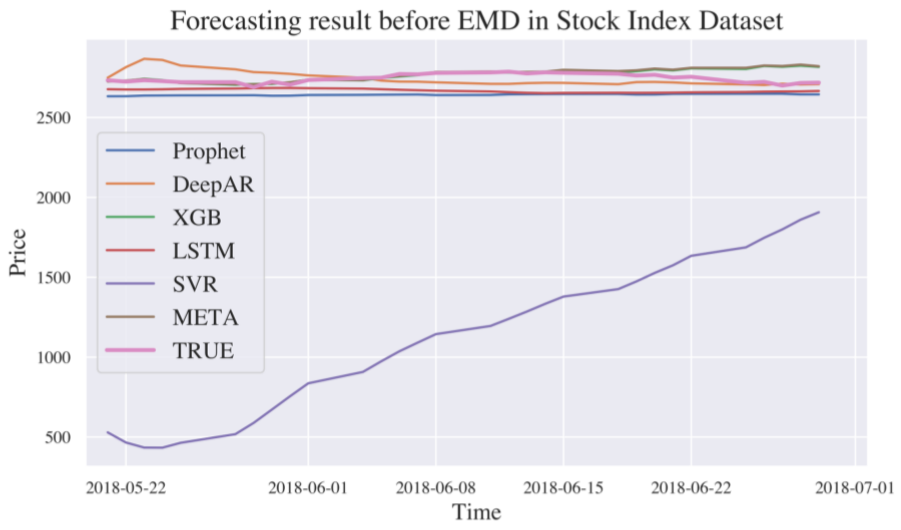

Fig 18 Forecasting result of methods before EMD in Stock Index Dataset

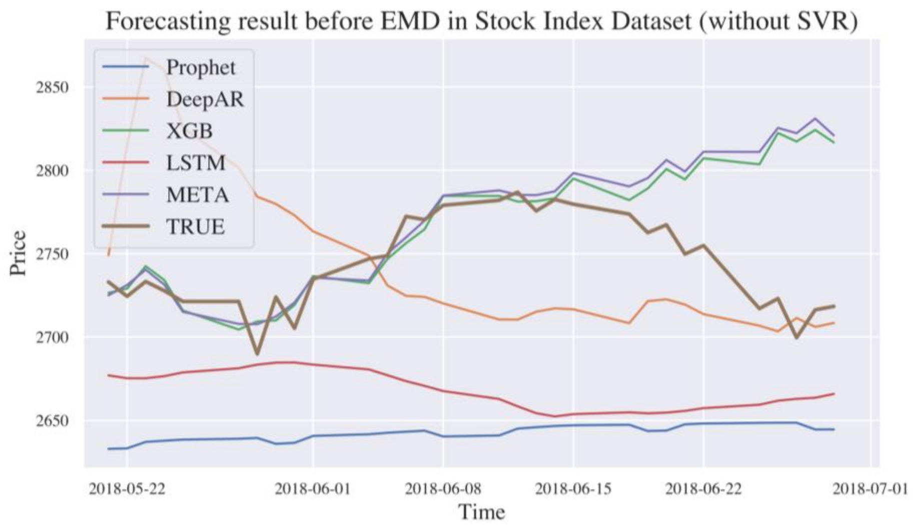

Fig 19 Forecasting result of methods (without SVR) before EMD in Stock Index Dataset

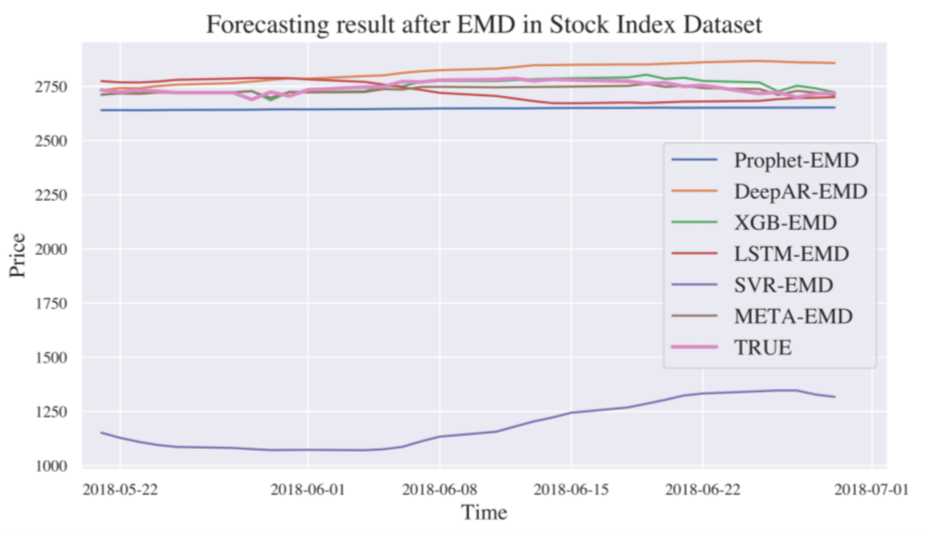

Fig 20 Forecasting result of methods after EMD in Stock Index Dataset

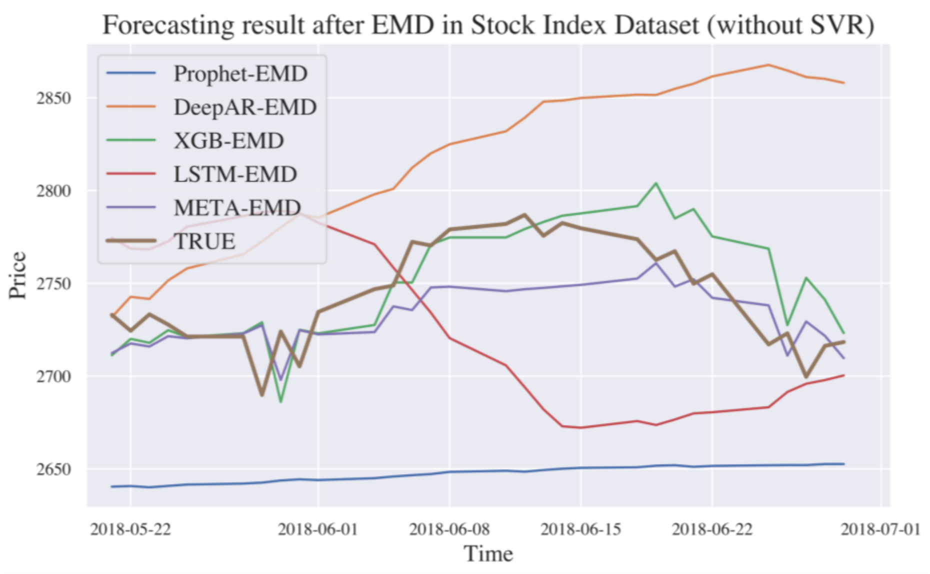

Fig 21 Forecasting result of methods (without SVR) after EMD in Stock Index Dataset

We plot the forecasting results before and after EMD in Stock Index Dataset in Fig 18 and Fig 20, respectively. But we find out results from SVR models are not close to the actual value, which compressed the plotting of other models' results. Thus, for better illustration, we plot the forecasting results without SVR models before and after EMD in Stock Index Dataset in Fig 19 and Fig 21, respectively.

From these figures, we can easily know that after EMD, the predictive prices from most models are closer to the real prices. Especially for XGBoost models and proposed meta-learning models, the trend is more correctly after EMD when compared to those before EMD. It goes up and drops down, which almost synchronized with the actual trend.

Moreover, we find out the results of our proposed methods are similar to those of XGBoost models, it proved that the meta-learning methods gave a large weight to XGBoost because of its superior performance.

6.1.3 Model-based Comparison

1) Prophet Model

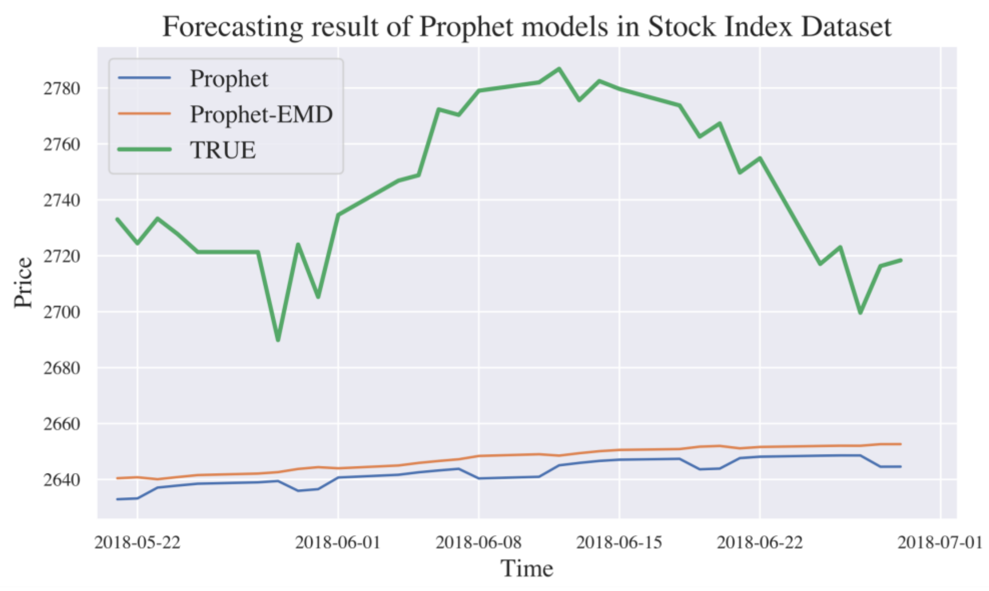

Fig 22 Forecasting result of Prophet models in Stock Index Dataset

As Fig 22 shown, after EMD, the result of Prophet method is more close to the real price, however, it still not well fit the real price and remain a huge gap between them.

2) DeepAR Model

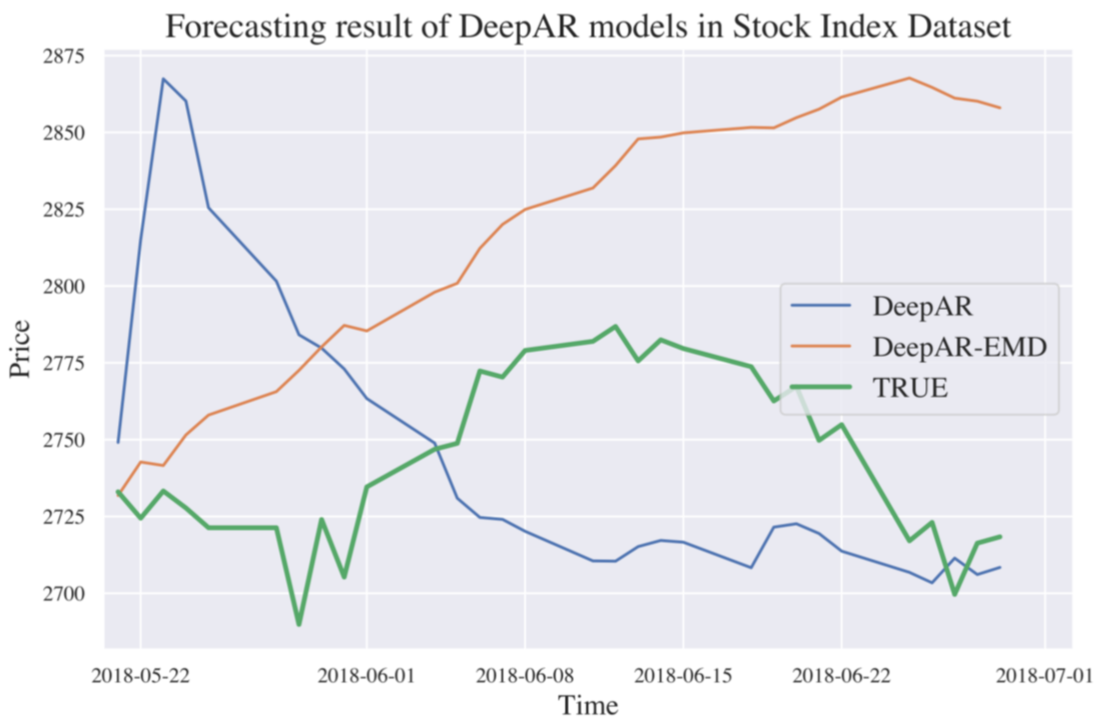

Fig 23 Forecasting result of DeepAR models in Stock Index Dataset

As Fig 23 shown, after EMD, the result of DeepAR model have an upward trend while those before EMD have a downward trend. However, both of them still not well fit the real price and remain a huge gap between predictive price and real price.

3) XGBoost Model

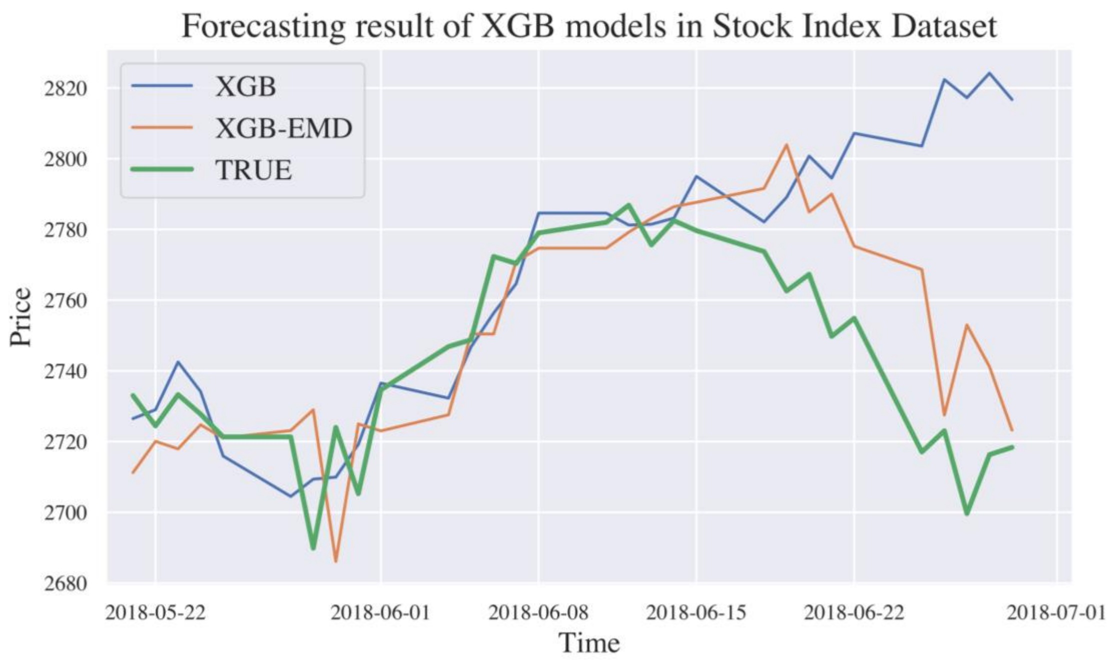

Fig 24 Forecasting result of XGBoost models in Stock Index Dataset

As Fig 24 shown, after EMD, the result of XGBoost model have similar trend with real price, moreover, the trend goes down, which is different to those before EMD. In overall, XGBoost models are well fit to the real price compared to other machine learning models. By decomposition, XGBoost model can well recognize the general trend of series.

4) LSTM Model

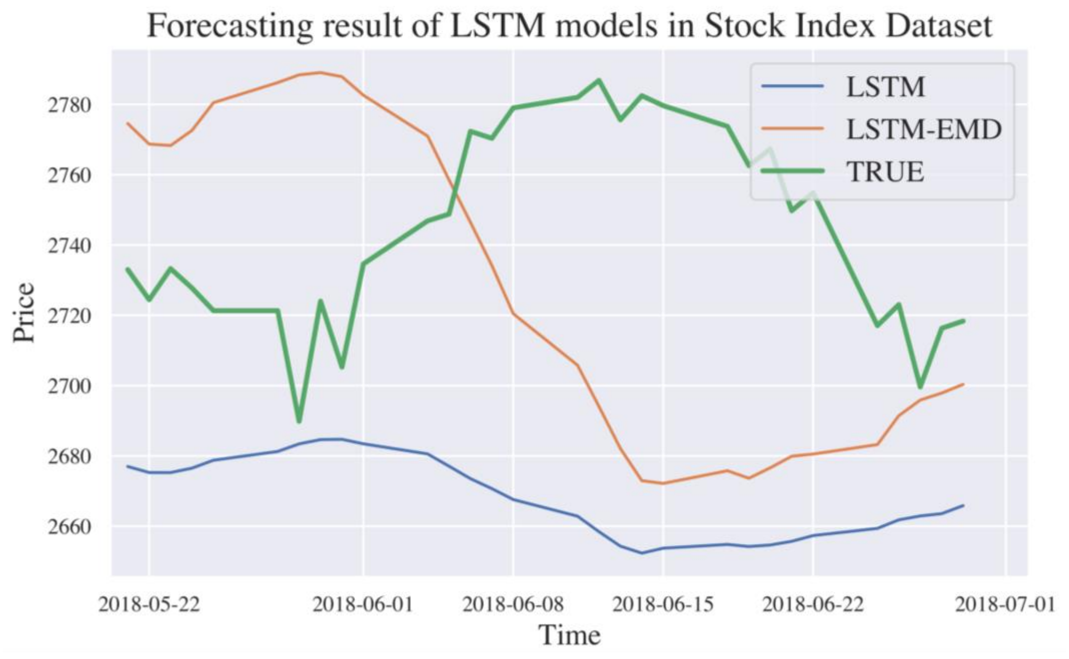

Fig 25 Forecasting result of LSTM models in Stock Index Dataset

As Fig 25 shown, after EMD, the result of LSTM model is closer to the real price when compared to model without EMD and it have larger decreasing range than that. However, both of them are not well fit the real price with a large gap between predictive price and real price.

5) SVR Model

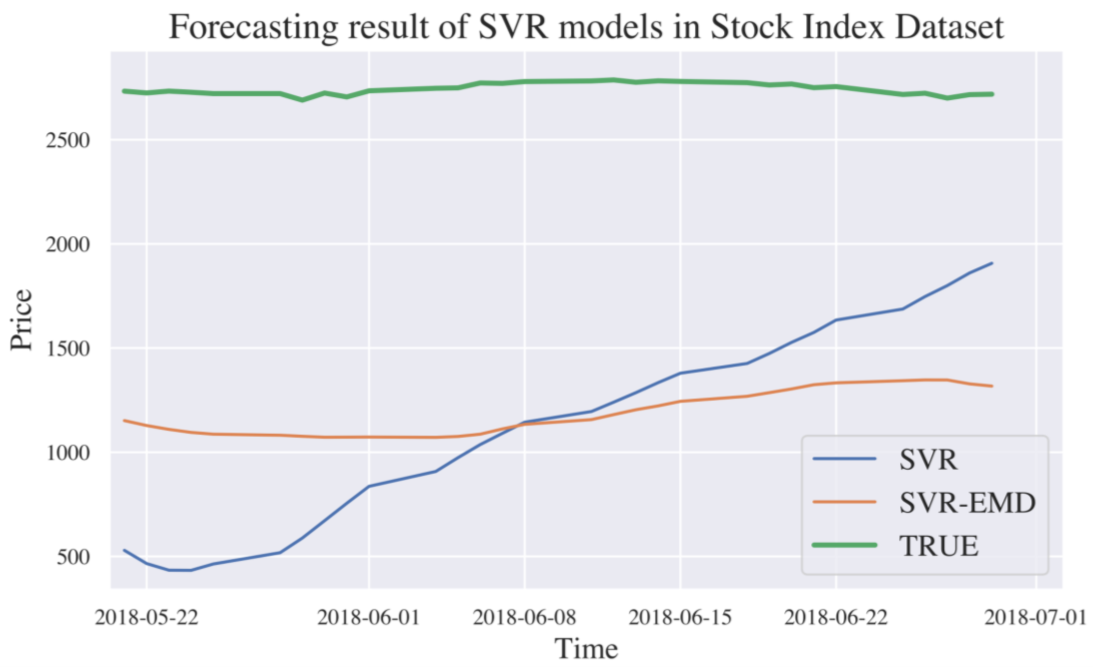

Fig 26 Forecasting result of SVR models in Stock Index Dataset

As Fig 26 shown, after EMD, the result of SVR model is closer to the real price when compared to model without EMD and it have milder increasing range than that. However, both of them are not well fit the real price with a large gap between predictive price and real price.

6) Meta Model

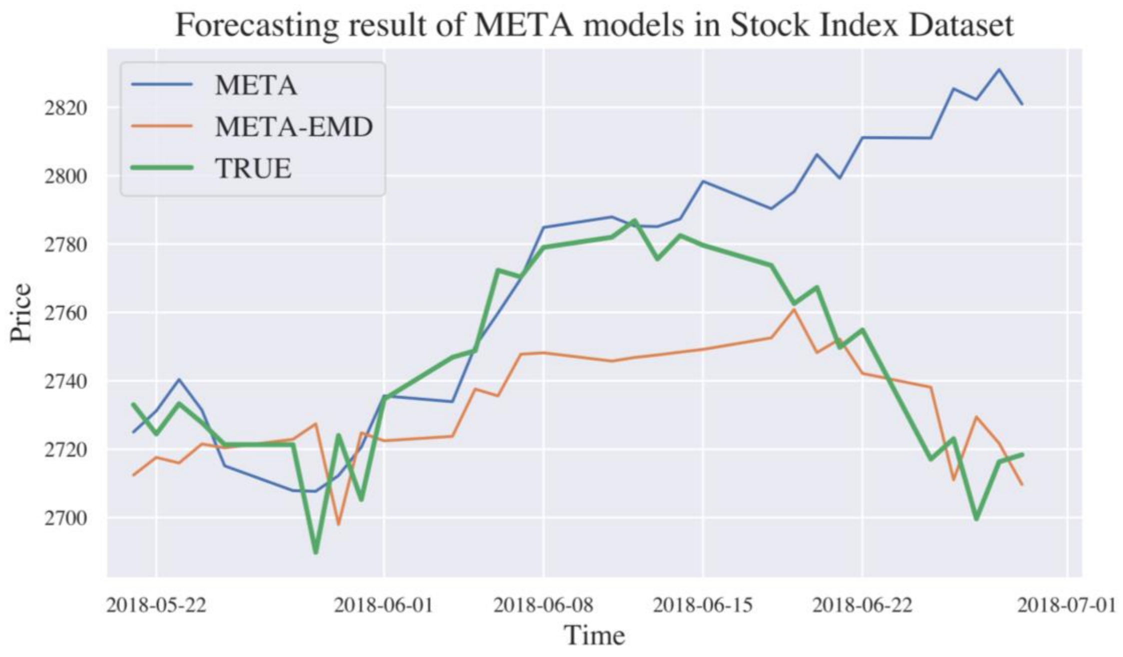

Fig 27 Forecasting result of META models in Stock Index Dataset

As Fig 27 shown, after EMD, the result of META model is closer to the real price when compared to model without EMD and its trend is more similar with the real price. It worth noting that the predictive price has a great similarity with the result of XGBoost, because META method assigned a large weight to the output of XGBoost. In overall, META model after EMD outperforms model without EMD which proves the effectiveness of proposed framework.

6.2 Exchange-rate Prediction

6.2.1 Forecasting Result

The forecasting results of Exchange-rate dataset are reported at Table 3, which shows as bellow.

Table 3 Exchange-rate Prediction results(in MSE and DA) of proposed framework and individual methods before EMD decomposition and after EMD decompositon.

	MSE		DA	
Method	Before EMD	After EMD	Before EMD	After EMD
DeepAR	1.509 x 10^-3	2.384 x 10^-3	34.5%	48.3%
Prophet	6.190 x 10^-3	5.097 x 10^-3	41.4%	37.9%
XGB	2.210 x 10^-5	9.400 x 10^-6	44.8%	55.2%
SVR	9.486 x 10^-2	4.921 x 10^-2	44.8%	58.6%
LSTM	4.331 x 10^-5	1.534x 10^-4	48.3%	41.4%
META	4.979 x 10^-5	1.002 x 10^-4	31.0%	58.6%

From Table 3, we can easily know our proposed framework (META after EMD) get the highest DA and the effect of EMD is significant that the DA of our framework achieved 58.6% which outperform those before EMD 89%, however, the MSE was increased from 0.0001 to 0.000049, nearly double increasement. Besides, during the EMD, there are different degrees of improvements among some models (Prophet, XGBoost and

SVR) in evaluation of MSE and the same situation happens in DA metric among DeepAR, XGBoost, SVR models. All these experiment results demonstrate our proposed framework works to improve the forecasting results.

6.2.2 Overall Comparison

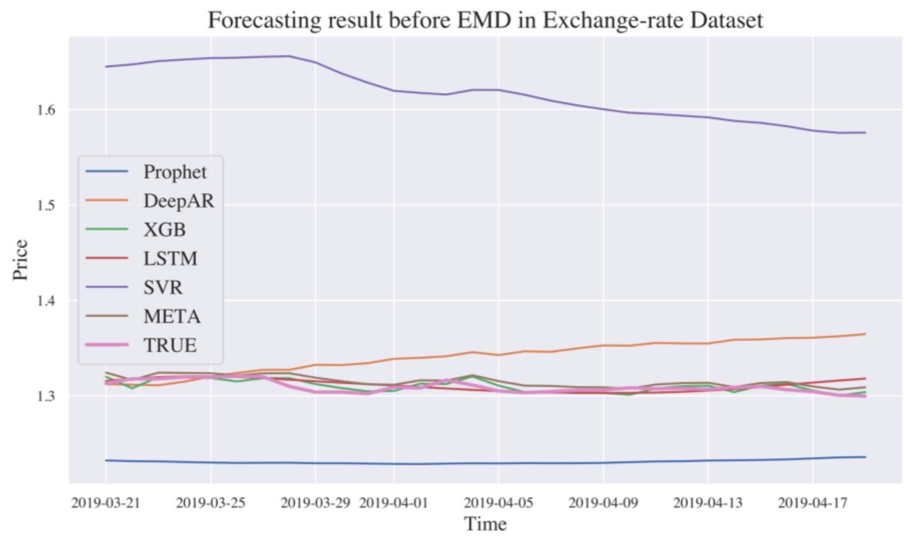

Fig 28 Forecasting result of methods before EMD in Exchange-rate Dataset

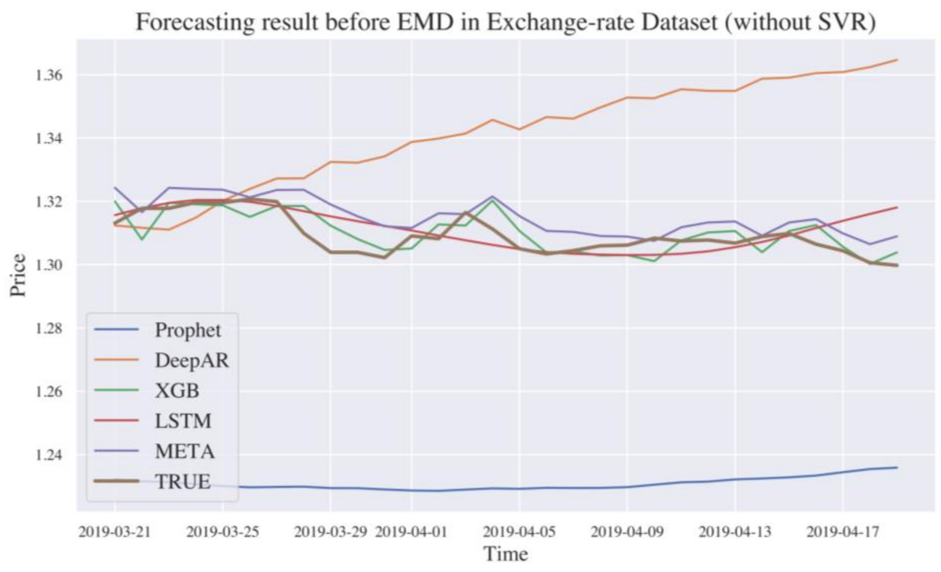

Fig 29 Forecasting result of methods (without SVR) before EMD in Exchange-rate Dataset

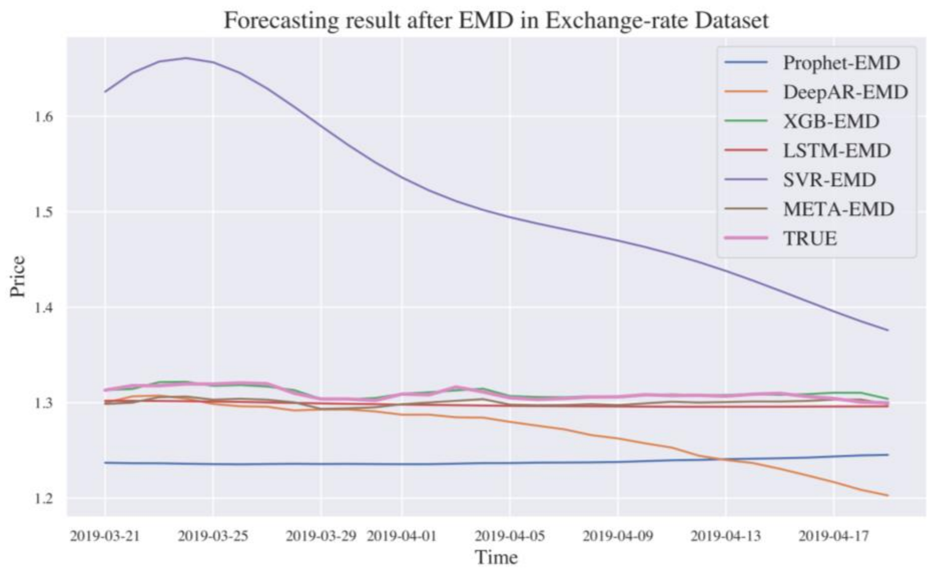

Fig 30 Forecasting result of methods after EMD in Exchange-rate Dataset

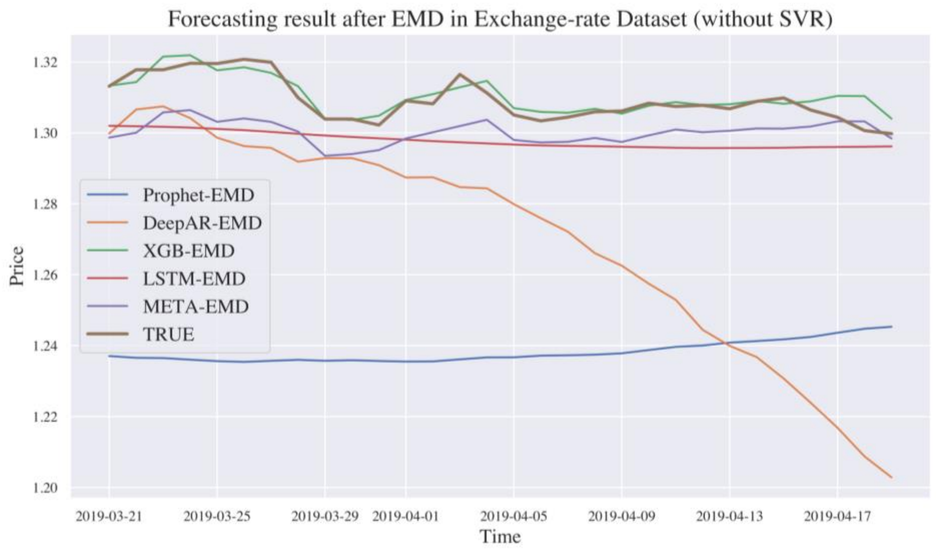

Fig 31 Forecasting result of methods (without SVR) after EMD in Exchange-rate Dataset

We plot the forecasting results before and after EMD in Exchange-rate Dataset in Fig 28 and Fig 30, respectively. But we find out results from SVR models are not close to the actual value, which compressed the plotting of other models' results. Thus, for better illustration, we plot the forecasting results without SVR models before and after EMD in Exchange-rate Dataset in Fig 29 and Fig 31, respectively.

From these figures, we can easily know that after EMD, the prediction from some models such as XGBoost and Prophet. However, not all models get improved after EMD, conversely, in some models, the gap between real price and prediction become larger. But the DA metric get improved after EMD, which means the framework still have some practical value in the real market.

6.2.3 Model-based Comparison

1) Prophet Model

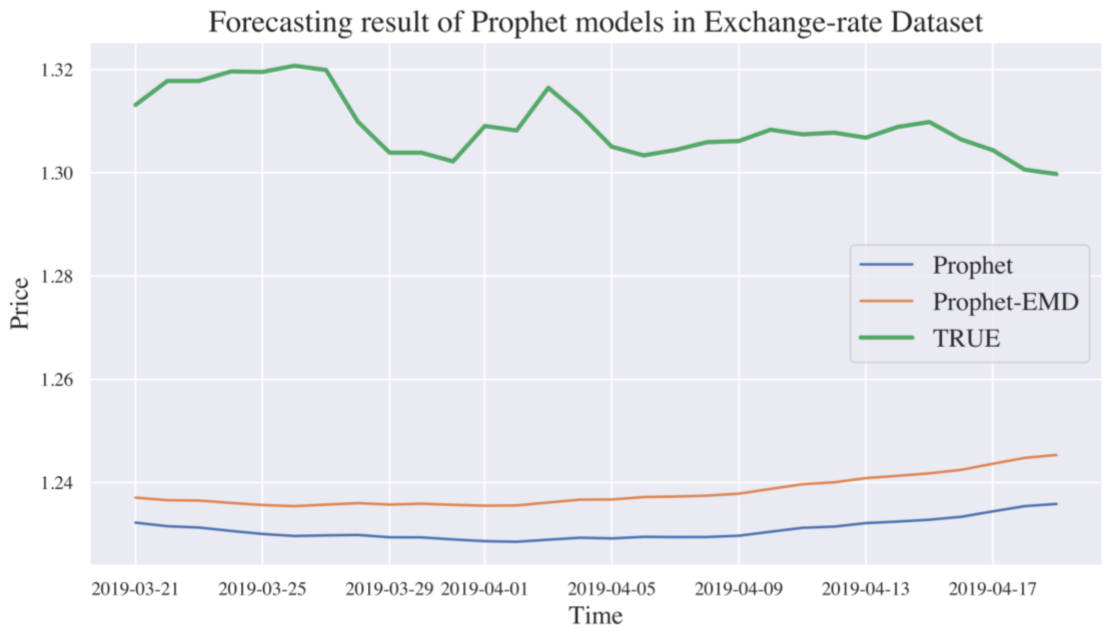

Fig 32 Forecasting result of Prophet models in Exchange-rate Dataset

From Fig 32, we can know, after EMD, the result of Prophet model is closer to the real price when compared to model without EMD. However, both of them are not well fit the real price with a large gap between predictive price and real price.

2) DeepAR Model

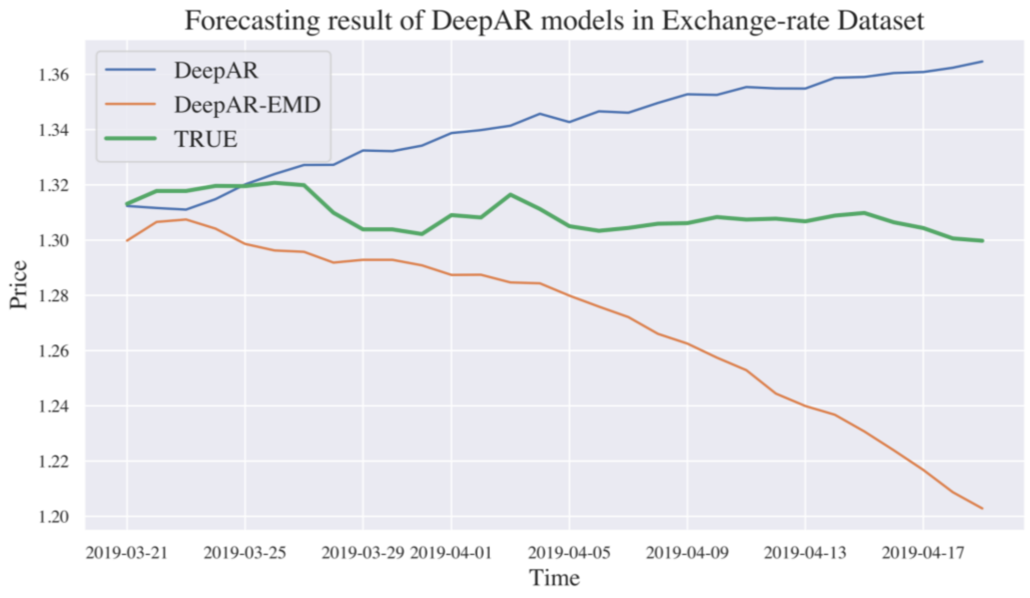

Fig 33 Forecasting result of DeepAR models in Exchange-rate Dataset

From Fig 33, we can know the result of DeepAR-EMD have a downward trend which those of DeepAR have an upward trend. However, both of them are not well fit the real price with a large gap between predictive price and real price.

3) XGBoost Model

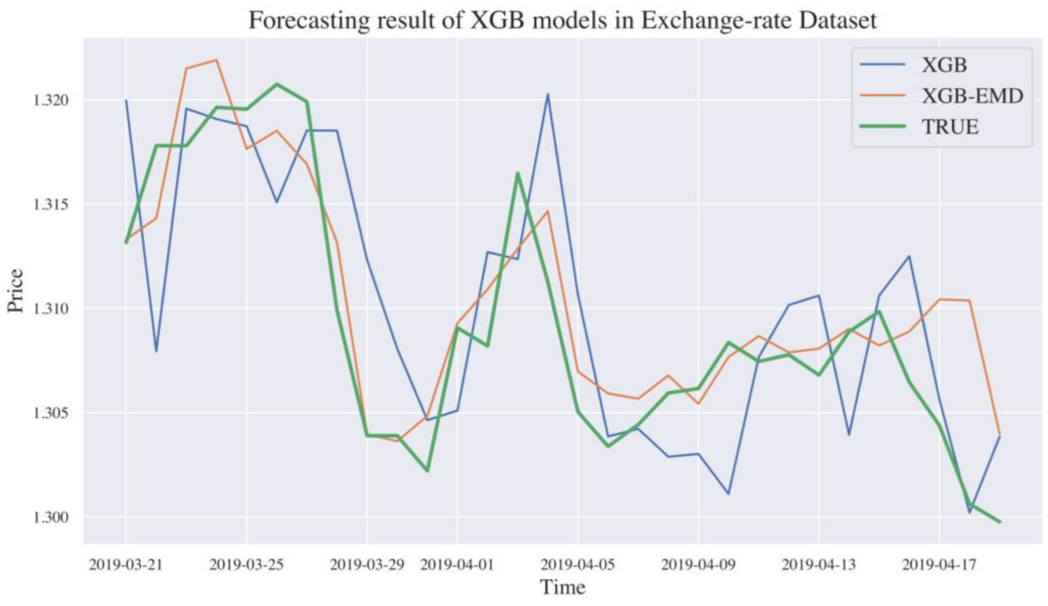

Fig 34 Forecasting result of XGBoost models in Exchange-rate Dataset

As Fig 34 shown, both of models before and after EMD fit the series well and have similar trend with real price. Compared to XGB, XGB-EMD performs better in Exchange-rate Dataset. In overall, XGBoost models are well fit to the real price compared to other machine learning models. By decomposition, XGBoost model can well recognize the general trend of series.

4) LSTM Model

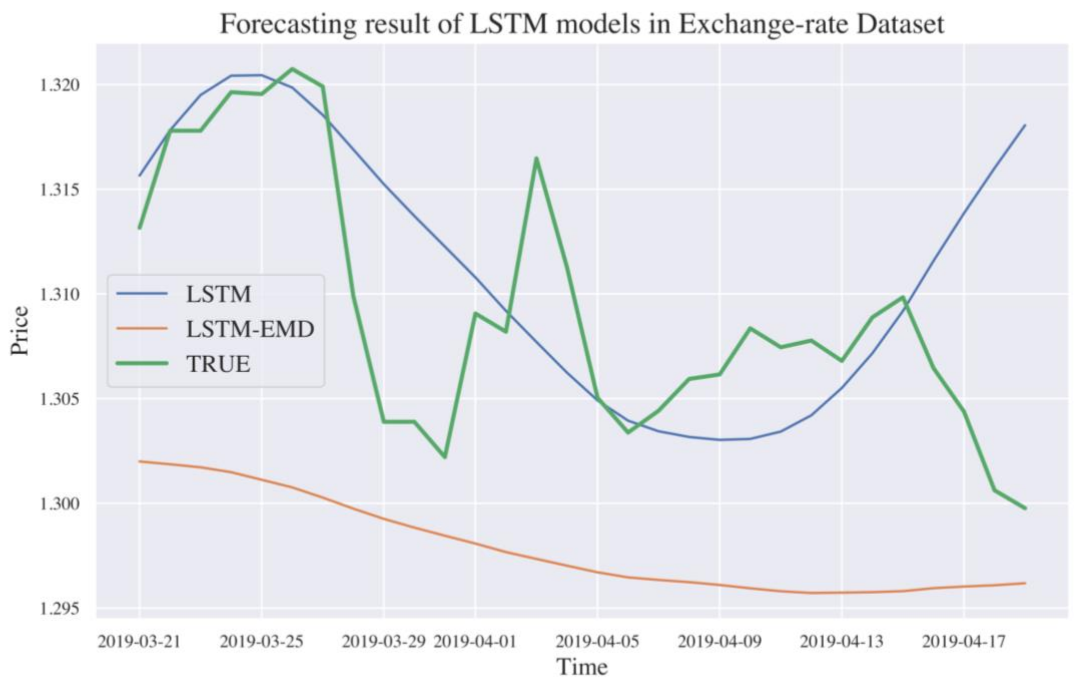

Fig 35 Forecasting result of LSTM models in Exchange-rate Dataset

As Fig 35 shown, before EMD, the result of LSTM model is closer to the real price when compared to those of model after EMD and the trend is quite similar with real price. However, both of them are not well fit the real price with a certain gap between predictive price and real price.

5) SVR Model

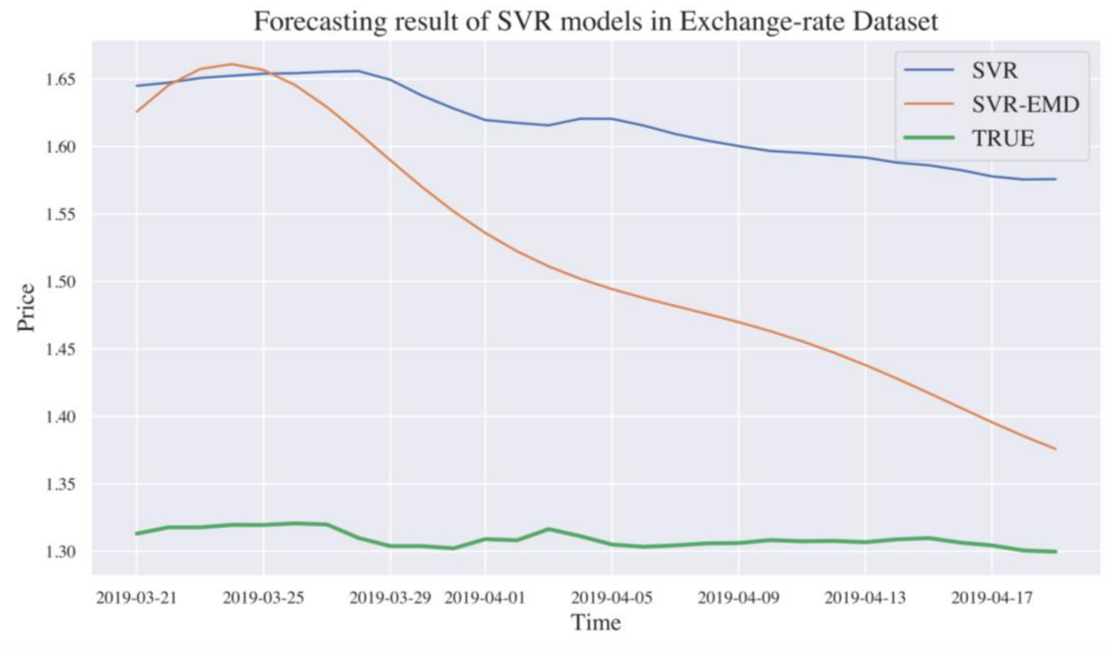

Fig 36 Forecasting result of SVR models in Exchange-rate Dataset

As Fig 36 shown, after EMD, the result of SVR model is closer to the real price when compared to model without EMD and it have acute decreasing range than that. However, both of them are not well fit the real price with a large gap between predictive price and real price.

6) Meta Model

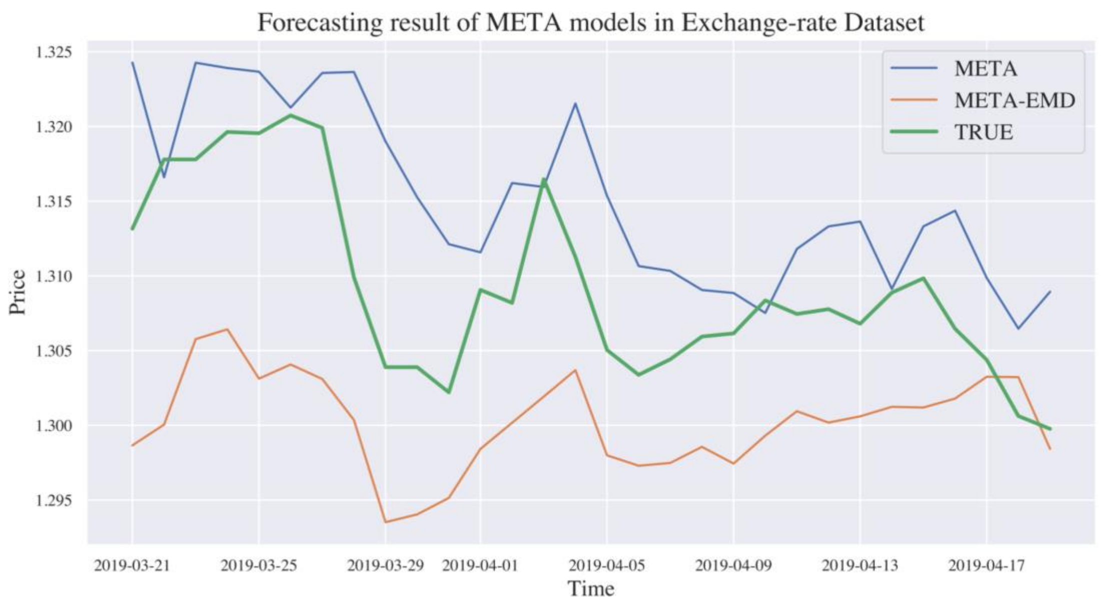

Fig 37 Forecasting result of META models in Exchange-rate Dataset

As Fig 37 shown, the result of meta model before EMD are closer to the real value compared to those of model without EMD. However, we can observe that the trend of model with EMD is slightly outperform those without EMD, it mainly due to that EMD is able to capture instant shocks from original series which are well fit by multiple model.

6.3 Oil Price Prediction

6.3.1 Forecasting Result

The forecasting results of Stock Index dataset are reported at Table 4, which shows as bellow.

Table 4 Oil Price Prediction results(in MSE and DA) of proposed framework and individual methods before EMD decomposition and after EMD decompositon.

Method	MSE		DA	
	Before EMD	After EMD	Before EMD	After EMD
DeepAR	1.533 x 10^3	1.390 x 10^3	37.9%	41.4%
Prophet	7.075 x 10^2	7.037 x 10^2	48.3%	51.7%
XGB	1.878 x 10^2	1.449 x 10^1	41.4%	65.5%
SVR	1.206 x 10^3	1.056 x 10^3	44.8%	58.6%
LSTM	1.597 x 10^3	1.700 x 10^3	58.6%	55.2%
META	2.483 x 10^2	2.382 x 10^1	41.4%	65.5%

From Table 4, we can easily know our proposed framework (META after EMD) get the second lowest MSE and the highest DA and the effect of EMD is significant that the MSE of our framework achieved 23.8 which outperform those before EMD 942%, in the meantime, DA increased from 41.4% to 65.5%, total increased 24.1% accuracy. Besides, during the EMD, there are different degrees of improvements among most models (DeepAR, Prophet, XGBoost and SVR) in evaluation of MSE and the same situation happens in DA metric among them. All these experiment results demonstrate our proposed framework works to improve the forecasting results.

6.3.2 Overall Comparison

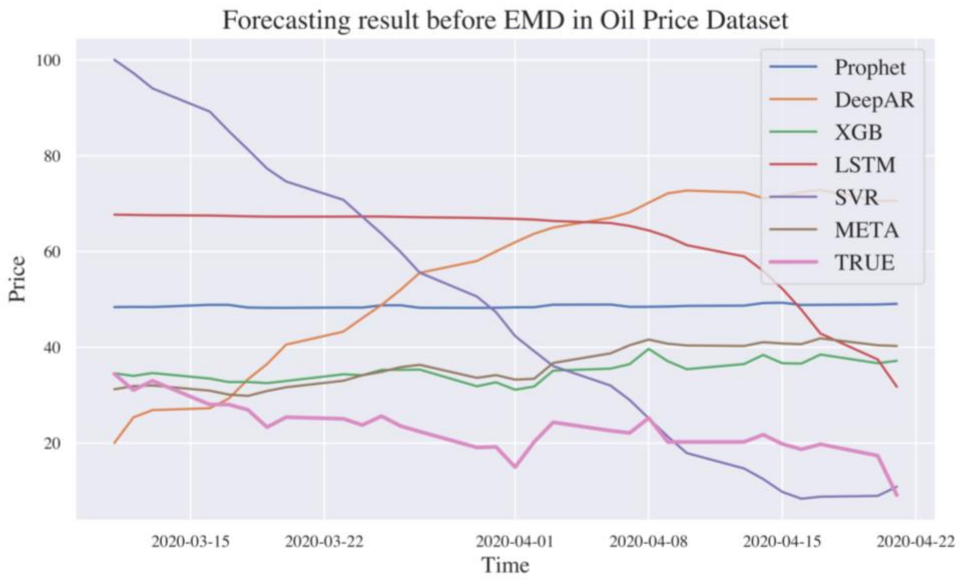

Fig 38 Forecasting result of methods before EMD in Oil Price Dataset

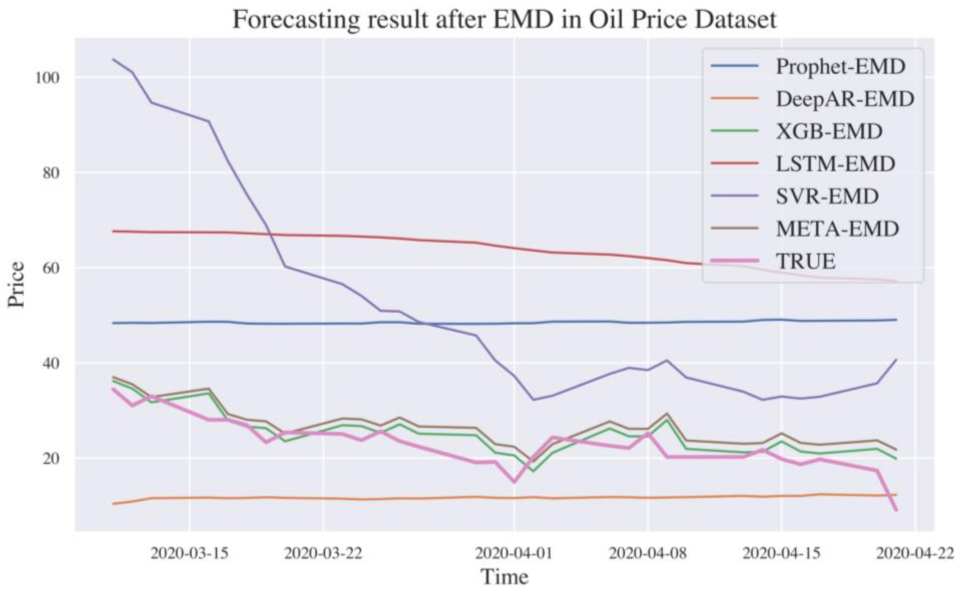

Fig 39 Forecasting result of methods after EMD in Oil Price Dataset

We plot the forecasting results before and after EMD Oil Price Dataset in Fig 38 and Fig 39, respectively.

From these figures, we can easily know that after EMD, the predictive prices from most models are closer to the real prices. Especially for XGBoost models and proposed meta-learning models, the trend is more correctly after EMD when compared to those before EMD.

Moreover, we find out the results of our proposed methods are similar to those of XGBoost models, it proved that the meta-learning methods gave a large weight to XGBoost because of its superior performance.

6.3.3 Model-based Comparison

1) Prophet Model

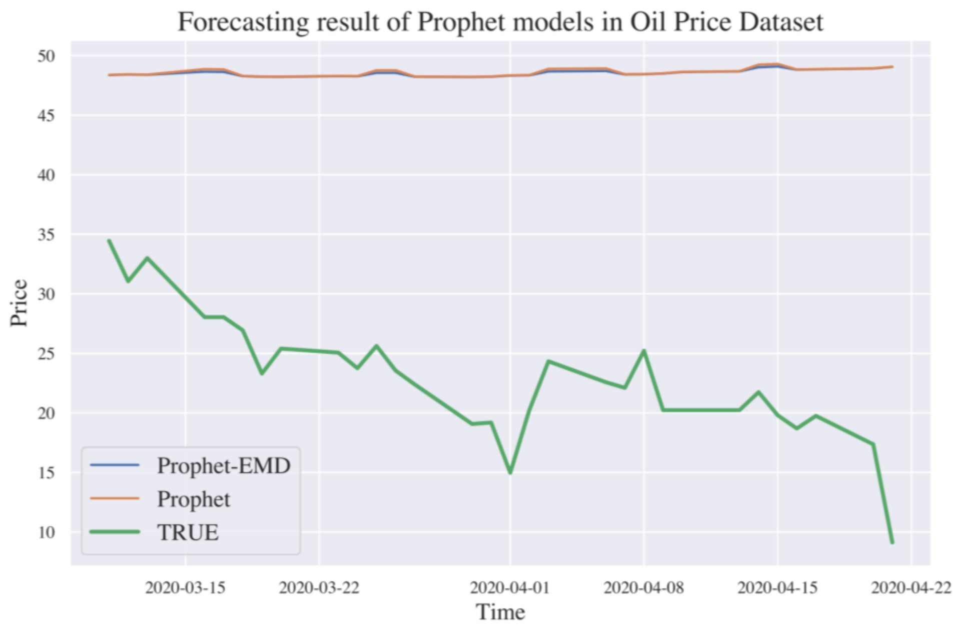

Fig 40 Forecasting result of Prophet models in Oil Price Dataset

From Fig 40, we can know, both of Prophet models with or without EMD appear similar results. However, both of them are not well fit the real price with a large gap between predictive price and real price.

2) DeepAR Model

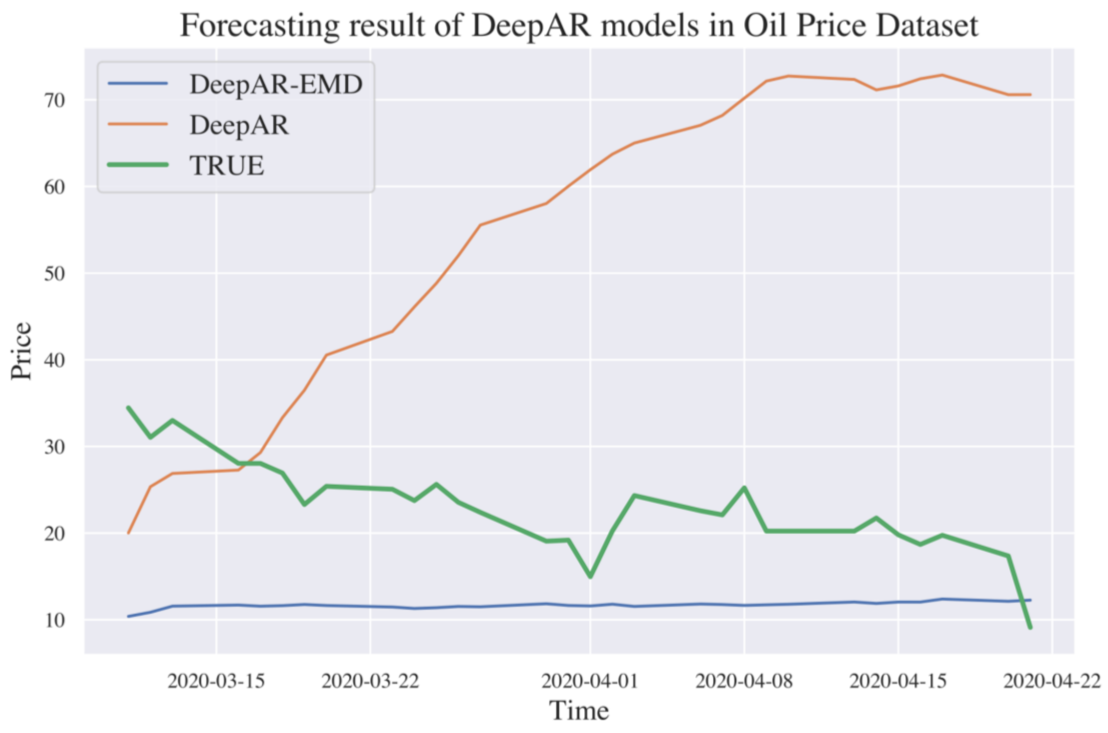

Fig 41 Forecasting result of DeepAR models in Oil Price Dataset

From Fig 41, we observe that the result of DeepAR model with EMD is closer to the real price when compared to those of model without EMD. However, both of them are not well fit the real price with a large gap between predictive price and real price.

3) XGBoost Model

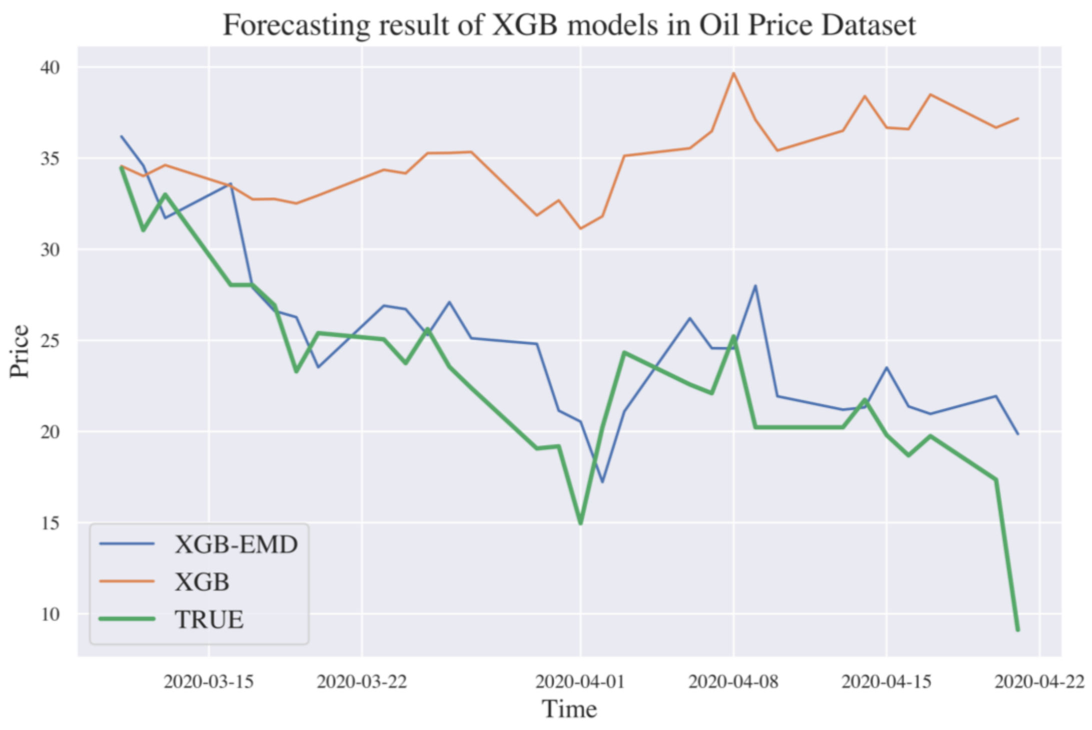

Fig 42 Forecasting result of XGBoost models in Oil Price Dataset

As Fig 42 shown, after EMD, the result of XGBoost model have similar trend with real price, moreover, the trend goes down, which is different to those before EMD. In overall, XGBoost models are well fit to the real price compared to other machine learning models. By decomposition, XGBoost model can well recognize the general trend of series.

4) LSTM Model

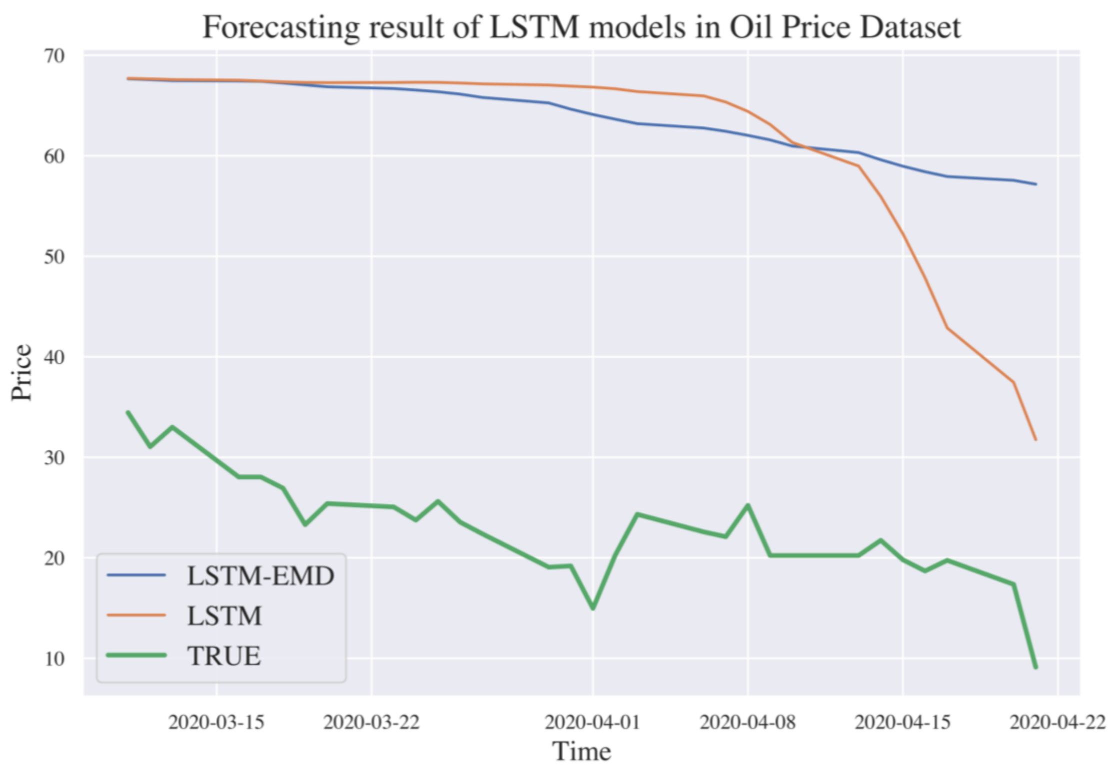

Fig 43 Forecasting result of LSTM models in Oil Price Dataset

As Fig 43 shown, both of models with or without EMD are not well fit the original series, which make the huge gap bewteen real price and predictive prices. However, we find the trend of prediction from model after EMD is more similar with those of real price when compared to model before EMD.

5) SVR Model

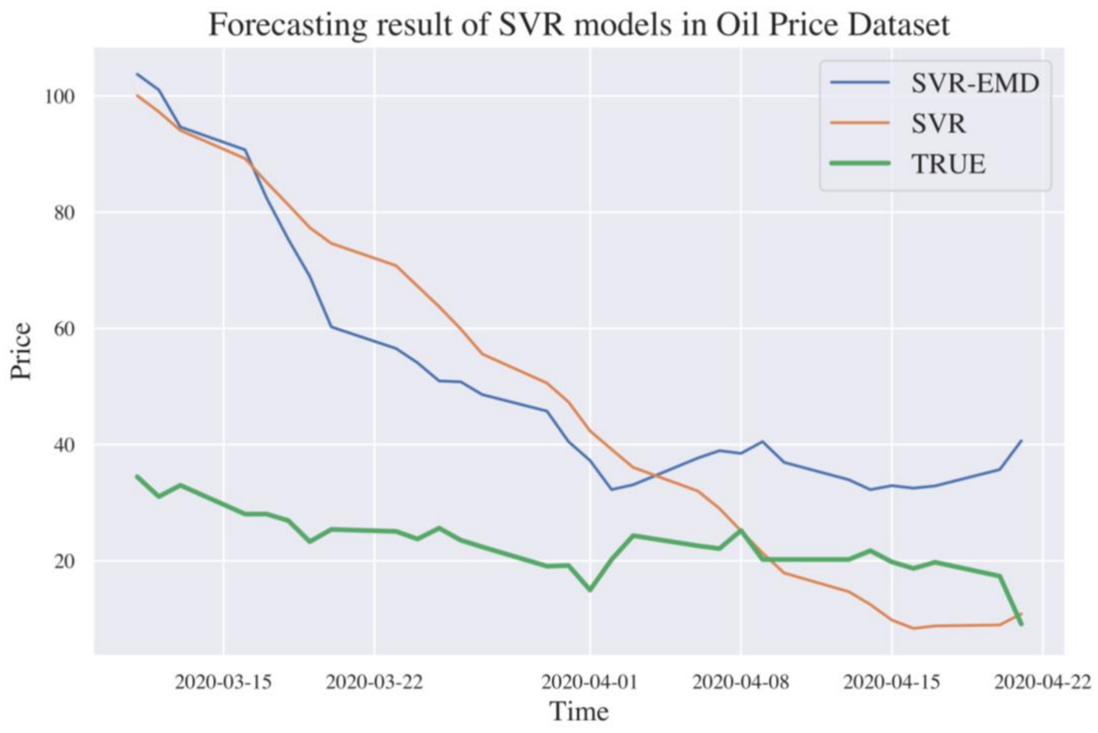

Fig 44 Forecasting result of SVR models in Oil Price Dataset

As Fig 44 shown, two predictions both have downward trend which is consistent with the real price. Compared to model without EMD, SVR-EMD's trend is more correspond with the real price. However, both of them are not well fit the real price with a large gap between predictive price and real price.

6) Meta Model

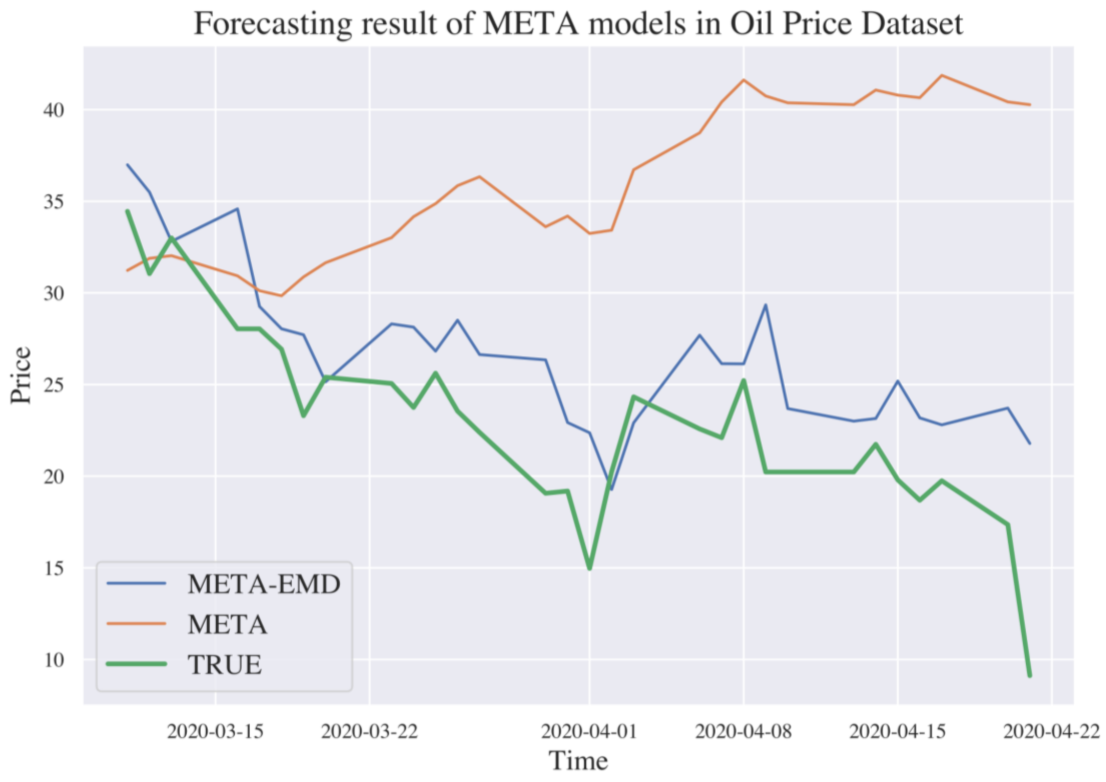

Fig 45 Forecasting result of META model in Oil Price Dataset

As Fig 45 shown, after EMD, the result of META model is closer to the real price when compared to model without EMD and its trend is more similar with the real price. It worth noting that the predictive price has a great similarity with the result of XGBoost, because META method assigned a large weight to the output of XGBoost. In overall, META model after EMD outperforms model without EMD which proves the effectiveness of proposed framework.

6.4 Result Analysis

From above subsections, we observed that our framework could improve the forecasting result from multiple base models, which means that the decomposition method do help to reduce the difficulty of forecasting original series. However, we found that not all model is fit well but the prediction of framework is good. This is because we employ the model ensemble mechanism to assign large weight to the model with good performance, which makes our framework more robust.

7. CONCLUSION

The prediction of final sequences always is an important topic for researchers and financial industry practitioners. Prediction can be hard when knowing no extra information such as market sentiment and some technical indicators. However, with development of machine learning model in time series forecasting and time series decomposition methods, it makes financial series forecasting being possible. In this dissertation, we proposed a framework based on EMD and model ensemble to slightly improve the forecasting accuracy. We adopt five different machine learning models, i.e. DeepAR, Prophet, SVR, LSTM and XGBoost, into our framework and use three type of financial data, i.e. stock index, exchange rate and commodity prices to evaluate the performance of proposed framework.

The general idea of our proposed framework is to decompose series with EMD into multiple IMFs and residual. Then, we use machine models to forecast these components and get the prediction of that after integrating outputs of models using weighted average method. After that, recompose series with these predictions then the forecasting of original series is obtained.

After conducting experiments on three datasets, we found our framework work in most scenarios. The prediction of S&P500 stock index, proposed method's MSE (511.6) outperforms that of the best individual model (XGBoost, 2110) while obtaining the best DA (a metric which can be regarded as the direction accuracy of next day price), 69%; In the prediction of exchange rate of USDGBP, best DA (65%) was obtained; In the prediction of Brent Oil Price, the proposed method achieve the best MSE (23.8) when compared with all individual models while getting the highest DA from 58.6% (from LSTM) to 65%.

In conclusion, by decomposing financial time series and combining different machine learning models, our framework can improve the forecasting result from the best base model. By employing model ensemble, our framework become more robust. And we believe it can provide useful prediction result when no extra information is knowing.

In future work, we could try more sophisticated model ensemble methods to improve the framework and collect more machine learning models into our framework.

REFERENCES

Agapie, A. (1997). *Forecasting the economic cycles based on an extension of the Holt-Winters model. A genetic algorithms approach.* Paper presented at the Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr).

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

Breiman, L. (1996a). Bagging predictors. *Machine learning, 24*(2), 123-140.

Breiman, L. (1996b). Stacked regressions. *Machine learning, 24*(1), 49-64.

Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5-32.

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

Cleveland, R. B., Cleveland, W. S., McRae, J. E., & Terpenning, I. J. (1990). STL: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics, 6*(1), 3-33.

Dagum, E. B. (1975). Seasonal factor forecasts from ARIMA models. *Bulletin of the International statistical Institute, 46*(3), 203-216.

Daubechies, I. (1992). *Ten lectures on wavelets* (Vol. 61): Siam.

Dhakal, D., Kandil, M., & Sharma, S. C. (1993). Causality between the money supply and share prices: a VAR investigation. *Quarterly Journal of Business and Economics*, 52-74.

Downs, G. W., & Rocke, D. M. (1983). Municipal budget forecasting with multivariate ARMA models. *Journal of Forecasting, 2*(4), 377-387.

Franses, P. H., & Van Dijk, D. (1996). Forecasting stock market volatility using (non-linear) Garch models. *Journal of Forecasting, 15*(3), 229-235.

Freung, Y., & Shapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J Comput Syst Sci, 55*, 119-139.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

Gordon, R. J., King, S. R., & Modigliani, F. (1982). The output cost of disinflation in traditional and vector autoregressive models. *Brookings Papers on Economic Activity, 1982*(1), 205-244.

Haar, A. (1909). *Zur theorie der orthogonalen funktionensysteme*: Georg-August-Universitat, Gottingen.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735-1780.

Huang, N. E., Shen, Z., Long, S. R., Wu, M. C., Shih, H. H., Zheng, Q., . . . Liu, H. H. (1998). The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London. Series A: mathematical, physical and engineering sciences, 454*(1971), 903-995.

Huang, N. E., Wu, M.-L. C., Long, S. R., Shen, S. S., Qu, W., Gloersen, P., & Fan, K. L. (2003). A confidence limit for the empirical mode decomposition and Hilbert spectral analysis. *Proceedings of the Royal Society of London. Series A: mathematical, physical and engineering sciences, 459*(2037), 2317-2345.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*: OTexts.

Issler, J. V. (1999). Estimating and forecasting the volatility of Brazilian finance series using ARCH models. *Brazilian review of econometrics, 19*(1), 5-56.

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

Kuncheva, L. I., & Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning, 51*(2), 181-207.

Lu, C.-J., Lee, T.-S., & Chiu, C.-C. (2009). Financial time series forecasting using independent component analysis and support vector regression. *Decision support systems, 47*(2), 115-125.

Mallat, S. (1999). *A wavelet tour of signal processing*: Elsevier.

Maravall, A., & Gómez, V. (1994). Program SEATS'Signal Extraction in Arima Time Series'. Instructions for the User.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2019). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing, 14*(3), 199-222.

Sollich, P., & Krogh, A. (1996). *Learning with ensembles: How overfitting can be useful.* Paper presented at the Advances in neural information processing systems.

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician, 72*(1), 37-45.

Tibshirani, R. J., & Efron, B. (1993). An introduction to the bootstrap. *Monographs on statistics and applied probability, 57*, 1-436.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks, 5*(2), 241-259.

Wu, Z., & Huang, N. E. (2009). Ensemble empirical mode decomposition: a noise-assisted data analysis method. *Advances in adaptive data analysis, 1*(01), 1-41.

Yan, H., & Ouyang, H. (2018). Financial time series prediction based on deep learning. *Wireless Personal Communications, 102*(2), 683-700.

Zhou, Y., Li, T., Shi, J., & Qian, Z. (2019). A CEEMDAN and XGBOOST-based approach to forecast crude oil prices. *Complexity, 2019*.