technische universität
dortmund

Bachelorarbeit

Automated Model Selection for Settlement Prediction in mechanized Tunneling processes

Donghui He

November 2018

Gutachter:
Prof. Dr. Katharina Morik
M.Eng. Amal Saadallah

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 8
http://www-ai.cs.uni-dortmund.de

Contents

Chapter 1

Introduction

1.1 Motivation and background

Since there is a specific target for each machine learning problem, we have to select the most appropriate model. In fact, some models are designed to only successfully solve some specific problems. For example, the convolutional neural network is good for image recognition. The recurrent neural network is also good for classification, but more suitable for sequence to-sequence identification.
As models trained on the same dataset show different performances, selecting and combining them is becoming more and more valuable. Model selection can improve the error tolerance when considering the models' performance. It also provides feedback about the strengths and weaknesses of a particular model under a particular condition. Therefore, our motivation is to build a framework that can select the best model under actual conditions. As the input data change, we could be able to get the best results from the model candidates.
The data analyzed in this thesis are time-series data, collected from a mechanized tunneling project. The data will be used to build an automated framework for forecasting model selection, and the selected models will be deemed as those most advisable to perform the prediction.

1.2 Problem description

Many machine learning models are able to perform time-series prediction. However, this performance partly depends on the data. When the input data vary over time, the accuracy of individual models may also change correspondingly.
In the time-varying environment, data characteristics and concepts may change over time. As a result, the predictive performance of forecasting models changes. For instance, ap-

propriate models for solving the task may become inaccurate. Our main goal is to select the most appropriate model over time steps for short-term settlement predictions.
The data of the experiment is described in subsection 1.2.1. For this data there are plenty of usable modules, both classic statistical ones or neural nets developed in recent years. Therefore, the approach to select those modules over time has direct influence on the final result. An automatic framework is to be built, so that the actually used model, which is selected from trained individuals, can adapt itself to the time-varying input.
The direction of current big data application is intellectualisation. However, automation should be the first aim. As the aspect of the design of our framework, it should be as automatic as possible. For that purpose, we need to organize and control our models. In our work, the framework should be able to select the most suitable models for different scenarios, and modules should be able to self-evolve along with timely usage.

1.2.1 Data description

Our data are extracted from a mechanized tunneling simulation. It is composed of a set of input-output data. The work contains 160 simulation scenarios; each of them contains exactly 64 steps. The input data include grouting pressure and support pressure data. The raw data of pressures are saved in a schema, in which rows denote time steps and columns denote numerous scenarios. Pressures have direct influence on point settlements, and they are exogenous variables in our experiment. The two kinds of pressure have values in the range from 10,000 to 22,000. To be an appropriate part of the model input, we normalized their values in the range from 0 to 1. The settlement of a point is measured at each time step and related to two given pressures. The output data are 160 *dat*files for 160 simulation scenarios, and in each *dat* file there are 154 columns representing 154 surface points and 64 rows for 64 time steps. The value in a cell indicates the settlement(displacement) of a surface point at a time step, which is an indication that a tunnel boring machine is excavating the soil.
The data preparation is a necessary procedure for the later work. Well organized data processing means data input that is adjustable according to requests and restrictions, e.g. memory limitation, time consuming and traceable checkpoints, etc.
The following data pre-processing is necessary:

- Each input sample is constructed as a scenario.
- The time-series for each input contains 64 tuples for 64 time steps; each tuple contains grouting and supporting pressures.
- Input and output data need to be correspondingly zipped together.
- Data should be normalised in a reasonable interval.

1.2.2 Approach to the problem

In this work different models (LSTM, GRU, MPL, ARIMA, VARX), both statistical models and neuronal networks and their ensembles, are built for time-series analysis. Among these models, two automatic model selection frameworks are built. A comparison will be established between the two methods, which are based on two approaches, namely an error-based and a meta-learning approach.

1.3 Aim of this work

The exerted pressures and point settlements are given as time-series. The surface points settlement are indicators of tunnel safety and sustainability. Therefore, it is important to provide accurate and reliable predictions of the settlement in mean-time, so that the input parameters can be adjusted.
Five individual models will be first constructed to predict the respective point settlements. Models trained on same data set mostly have different performances, so we will combine individual models to improve the forecasting accuracy and avoid extreme situations. However, such ensembles are not absolutely better than individual models, as ensembles are more complex. We will build ensembles with all combination possibilities, setting a given number of ensemble members.
The aim is to build an automated forecasting model selection framework for these time-series data. The proposed framework will be applied to select a candidate model at each time step to perform prediction tasks. Model selection approaches are built to estimate the performance of models by outputting the estimated error or rank of its candidate models, and the model with the best performance in the framework is selected. Prediction will be based on the actual observation and previous performance of the associated model. There will be two model selection approaches, error-based and meta-learning. They will estimate which model would be best to use for predictions under the current circumstances.
The selected ensembles, using error-based dynamic model selection and meta-learning approaches, will be also compared with each other, as well as with individual models from different aspects. It is expected that the model selection approach will be a promising way to improve the prediction accuracy of the surface settlements.

1.4 Thesis structure

The main thesis consists of four parts. The first part is about our motivation and aim of this work. In next part – chapter 2, time-series basics are introduced, as well as the forecasting models – in chapter 2.3, 2.4, 3 details of .individual models are contained. In chapter 3, ensemble model and the two approaches for model selection are detailed. In the last part, chapter 4, we present our experiments and implementation details, in order to draw a transparent conclusion, taking the aspects into account. And finally, chapter 5 provides a summary and outlook of our work. More results and graphs are included in the Appendix.

Chapter 2

Time-Series Modeling

Our main goal is to build an automatic model selection framework for time-series forecasting models. Time-series analysis is an important part of data analysis that involves time-dependent data. In recent years, neural networks are highlighted for solving problems in many areas. More and more, time-series studies that used to be solved with statistical models are nowadays approached with neural networks. In later experiments, we will show that networks are quite suitable for forecasting time-series data. In this section, the concept and the basic characteristics of time-series will be detailed.

2.1 Time-series basics and forecasting models

Time-series analysis is the analysis of a series of data-points that are listed in time order. The observation of data-points can be made continuously or at specific time. We can express a series as x1,x2,x3,...,xT or xt where t = 1,2,3,...,T. xt is the observation of data in time t, the sequence of xt, x1,x2,x3,...,xT , which indexed by time, is called time-series.

2.1.1 Definition. Time-series analysis is the general expression for techniques that extract useful information such as statistic features and other characteristics from time-series data.
In time-series predictive analytics, features of time-series are extracted for forecasting. Time-series forecasting is normally based on previously made observations, to predict future values of time-series. In our experiment, the data of point settlements and pressures are both in the form of time-series. Only the point settlements are the observed and are to be predicted; the pressures are exogenous.

Notation. Endogenous variables Endogenous variables or primary series are a classification of variables generated by a model, and also the original data.

Notation. Exogenous variables covariates is used for indicating arbitrary external conditions that affect endogenous variables without being affected by them in turn. Exogenous variables have observations at the same time steps as the original series.
A time-series can be decomposed into three components, namely the factors of changes. The components of time-series are listed in the following:

Seasonality are short-term movements in time-series that are reasonably stable with respect to timing. They arise from systematic, calendar-related influences such as festivals, natural conditions, and social and administrative procedures.

Irregularity is sometimes called residual or random components. It remains after removing the seasonal and trend component of a time-series. It means sudden changes occurring in a time-series which are neither systematic nor predictable, such as earthquakes or strikes.

Trend is the main component of time-series, and indicates long-term effects without calendar-related events and irregularities. It represents the underlying level. Usually socio-economic and political factors bring about this effect, such as population growth or inflation, which shot the growth or decline in time-series over a long period. The decomposition models are typically in additive or multiplicative forms. If the seasonality and irregularity of a time-series are independent of the trend, the additive model is appropriate for the decomposition:

$$O_t = T_t + S_t + I_t$$

where O_t is the observed time-series, S_t seasonality T_t trend I_t is the irregularity. However, the multiplicative mode is appropriate when both the seasonal and irregular components change along with the trend component. The observed time-series is the result of multiplying the three components in the multiplicative model, i.e. in following formula:

$$O_t = T_t \times S_t \times I_t$$

When a time-series analysis model is given for specific time-series data, those data are assumed to have stationarity properties.

Stationarity is a property of time-series. A time-series is classified as stationary series when its mean, variance and auto-correlation structure do not change over time.

Time-series with trends or seasonality are not stationary, because fluctuations brought by trend and season will affect their values. Hence, it is necessary to analyse the trends before building the time-series model. There are multiple ways to stationarize a non-stationary series once we know its trends, cycles and seasonality. Detrending can model the identified trend. Once modelled, the trend component can simply be removed from the time-series. The following describes two commonly applied methods for stationarizing:

Detrending by differencing : We can difference the data. Here we only discuss the additive model, the multiplicative model also applies a similar formula:

$$Y_i = X_i - X_{i-1}$$

where X_i is the given time-series set, X_{i-1} the given set in the last time step and Y_i is the newly created series. This has the effect of removing a trend from a time-series dataset. Although the difference can be applied more than once, one difference is generally sufficient.

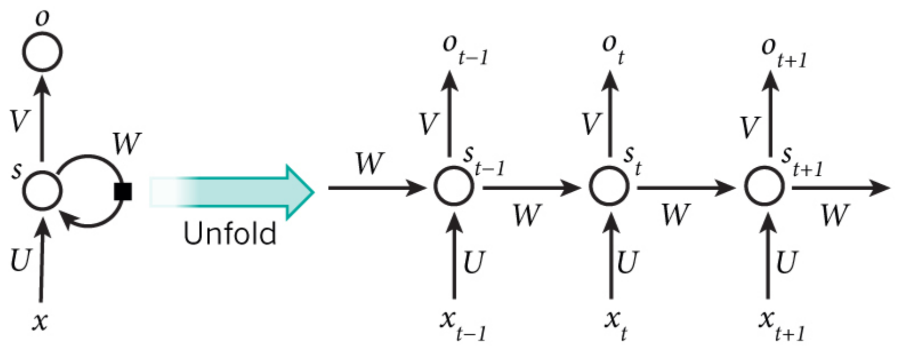
Detrending by model fitting : The trend component of a time-series can also be removed when its predictions are taken as the trend line, as in the following formula:

$$R_i = X_i - \hat{X}_i$$

where $\hat{X}_i$ is the predictions. After the fit of the model, the residuals R_i are a detrended form of the time-series. It can also be applied to other non-linear models.

2.2 Time-series forecasting models

There are different powerful analysis tools for time-series in both statistical and neural networks. Models like autoregressive integrated moving average (ARIMA), vector autoregressive with exogenous variables (VAR-X), and support vector regression (SVR) are used in time-series forecasting with different performance. ARIMA and VARX are applied in our experiment. ARIMA is a general univariate model, based on the assumption that the time-series being forecasted are linear and stationary[8]. VARX is a multivariate time-series mode and suitable for influence of lag for both endogenous and exogenous variables. More details about ARIMA and VARX can be found in 2.3.
Neural networks actually perform well, not only for classification tasks but also regression analysis, like feed-forward neuronal networks, recurrent neural networks (RNN) with long-short-term memory cells (LSTM) and gated recurrent cells (GRU) or recently emerged convolutional LSTM recurrent network cells [24].
In an artificial neural network, RNN is a connectionist model containing a self-connected hidden layer [9]. Connections of RNN form a directed graph along a sequence. The following shows the structure of a basic RNN. RNNs are called recurrent because they perform the same computing process for every element of a sequence. Each output is actually affected by the RNN status, which depends on the previous inputs. RNNs impressively describe the dependence of time-series,



Source:
`http://www.wildml.com/wp-content/uploads/2015/09/rnn.jpg`

Figure 2.1: A recurrent neural network and the unfolding in time of the computation

where the sequence graph after "unfolding" is more comprehensible to explain clearly how a recurrent cell works, because in practice the following information is not recycled infinite times. Unfolding shows the RNN model as a directed sequence. x_t is the input at time step t. o_t is the output at step t, it has value of $o_t = softmax(V_{x_t})$. s_t is the hidden state at time step t, indicating the "memory" from previous steps, so is $s_t = f(U_{x_t}+W_{s_{t-1}})$. Method f indicates the hidden unit structure, which depends on what kind of hidden cell applies. LSTM and GRU cells have more complex gate units V, U, W,and will be explained in chapter 2.4.

Beside the above-mentioned kinds of models, we also found that their ensemble models are usually helpful in improving the performance of prediction tasks. In our thesis, we use selected models from five different candidate models and their compositions (also called ensembles).

We also used ensemble of individuals model to perform the prediction task. In [22], the ensemble of classification models performed with more both efficient and accuracy for mining concept-drifting data streams. The individual models are weighted according the expected classification accuracy of the test data. A hybrid ARIMA and neural network model was built in [26], to take advantage of the unique strength of the ARIMA and ANN models. Three well-known data sets, Wolf's sunspot data, Canadian lynx data, and British pound=US dollar exchange rate data were used in the experiment, and the result shows that the combined model is a promising approach to strengthen model performance.

The ensemble construction of our work is detailed in chapter 3.1. Ensembles of our work are not only applied to improve prediction accuracy, they also extend the set of candidates for the selection of most appropriate model.

2.3 Statistical models

Two statistical models will be built for each surface point. For each prediction, the two applied pressures and the settlements in the actual sliding window will be included in the model as exogenous variables. Because of the limited number of steps in each scenario, it is reasonable to concatenate the diverse scenarios together for training.

2.3.1 ARIMA

The ARIMA (autoregressive integrated moving average) model is also known as the Box-Jenkins method. ARIMA is one of the popular linear models for time-series forecasting. With ARIMA, non-stationary time-series are decomposed into a permanent and transitory (cyclical) component, allowing both components to be stochastic [1]. It is also used in many practical problems, such as the prediction of next-day electricity prices [6].
ARIMA can be viewed as a filter that separates the noise from the input and then predicts the future. The order (p, q) of ARIMA is important for the accuracy of prediction. ARIMA generates time-series as the following formula:

$$R_t = \kappa_0 + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \varepsilon_t - \kappa_1 \varepsilon_{t-1} - \kappa_2 \varepsilon_{t-2} - \cdots - \kappa_q \varepsilon_{t-q}$$

Where R_t and ε_t are the target value and random error at time step t.
The autoregressive part of ARIMA evolves itself with the historical data, which are called lags.
The Autoregressive component p indicates the order of the autoregressive model and the number of time lags. When $p = 0$, it means that there is no auto-correlation in the series. Also, if $p = 1$, the auto-correlation lag of the time-series is set to 1.
The integrated component d denotes times of applied difference operators, which are called moving integrated terms. Applying the difference operator to a non-stationary series x_t once:

$$\nabla x_t = x_t - x_{t-1} = w_t$$

where w_t is a stationary series. If d is set to 0, we say the time-series data is stationary, for non-stationary time-series $d = 1$ or $d = 2$ is in usual case sufficient.
The moving average component order q indicates the number of lagged forecast errors. The random errors ε_t is white noise when $E(w_t) = 0$ and constant variance equal to σ^2 and no serial correlation(i.e. $Cor(\varepsilon_i, \varepsilon_j) = 0, \forall i \neq j$).
Through both the auto-correlation and partial auto-correlation, p, q and weights are inferable from previous time-series. In this work, it could be very helpful when the data are concatenated (normally data in 64 steps are a relatively short time-series), because the frequencies of the concatenated data should be easy to find out.

2.3.2 VARX

Another statistical model in our work is the VARX (vector autoregressive with exogenous variables) model, which has been used in many works to perform time-series prediction tasks, such as [7] in 2000 and [17] in 2005. It is expected to give time-series techniques more credibility, as stated by Dekimpe and Hanssens in [7]. Therefore, relevant constraints are needed for determining the VAR coefficients, which can lead to poor forecasts (large forecast intervals) [17]. Vector autoregressive (VAR) process is a widely used method in time-series forecasting to estimate the relation between the current values of a multivariate time-series and their lagged values. There are better ways to extract information for insignificant coefficients; the consequences of estimating the VAR coefficients are to be considered to avoid the uncertainty of expansive estimates [17].
The VARX model is a variant of the stationary VAR model; the latter can be used to analyse the linear interdependencies among multiple time-series based on its own lagged values, as shown in the following equation:

$$y_t = \delta + \sum_{i=1}^{p} A_i y_{t-i} + \varepsilon_t \tag{2.1}$$

where y is the endogenous variable indexed with step. Equation (2.1) can be written in following multivariate linear model:

$$y = (X \otimes I_k)\beta + e$$

where $y = vec(y_1, y_2, ..., y_T)$, $X = vec(X_0, X_1, ..., X_{T-1})$, $X_t = vec(1, y'_t, y'_{t-1}, ..., y'_{t-p+1})$, $\beta = vec(\delta, A_1, \ldots, A_T)$ and e is residual error vectors with $e = vec(\varepsilon_1, \varepsilon_2, ..., \varepsilon_T)$.
To obtain parameters β, the conditional least squares estimator is used:

$$\hat{\beta} = ((X'X)^{-1}X' \otimes I_k)y$$

In our experiment, the VARX model is applied to manually add exogenous variables to the estimation process.
VARX is the VAR(vector autoregressive) model with exogenous variables, as elaborated in [2] :

$$y_t = \delta + A_1 y_{t-1} + ... + A_p y_{t-p} + B_1 x_{t-1} + ... + B_q x_{t-p} + \varepsilon_t$$

where $Y_t \in R^k, X_t \in R^m, U_t \in R^k$ are vectors. X_t are vectors of exogenous variables U_t is the vector of errors, and $A_0, ..., A_p, B_0, ..., B_q$ are matrices shaped $k \times k, k \times m$ respectively. Because the part of exogenous variables is already known, we can also transfer it to a multivariate linear model:

$$y = (X \otimes I_k)\beta + e$$

where $y = vec(y_1, y_2, ..., y_T)$, $X = vec(X_0, X_1, ..., X_{T-1})$,
$X_t = vec(1, y'_{t-1}, \ldots, y'_{t-p}, x'_{t-1}, \ldots, x'_{t-p})$, $\beta = vec(\delta, A_1, \ldots, A_p, B_1, \ldots, B_q)$ and e is

residual error vector with $e = vec(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_T)$. Therefore, the conditional least squares estimator of β is also in the form:

$$\hat{\beta} = ((X'X)^{-1}X' \otimes I_k)y$$

As our thesis is weighted more on the framework than individuals, for more details about VARX please see Sergio etc. 2012 [19].

2.4 Neural network models

Compared to the previously mentioned ensemble members, the neural network has attracted much attention in recent years, and has also achieved great success in many areas. The recurrent neuronal network(RNN) is well known by its long-term-dependency (section 2) for processing a sequence of data. The simplest recurrent neuronal network has only one perceptron in it. This kind of network is special, in that it not only produces a simple output, but that this output also reverses itself as the input for the next step. This special structure leads data to follow recurrently. This structure is widely used in the theory of control and regulation as dealing with time sequence signal. Thus, the first input may have a long-term influence on the futrue outputs . At this point, it can simulate data in the form of sequence and so as time-series analyses. The RNN had shown incredibly good performance in handwriting [9] and speech recognition[21].

2.4.1 MLP

The MLP (Multilayer Perceptrons) is a feed-forward artificial neural network consisting of at least three layers of perceptrons. Each perceptron is actually a function with input, which can be a multidimensional array, and output, which is usually a one-dimensional number. The perceptron represents a function called activation function, which maps the input with its own bias, generally in intervals [0,1]. The training phrase of MLP uses a supervised learning method called backpropagation. The general purpose of MLP is binary classification and non-linear separation. An example of a MLP structure is shown as following figure 2.2.

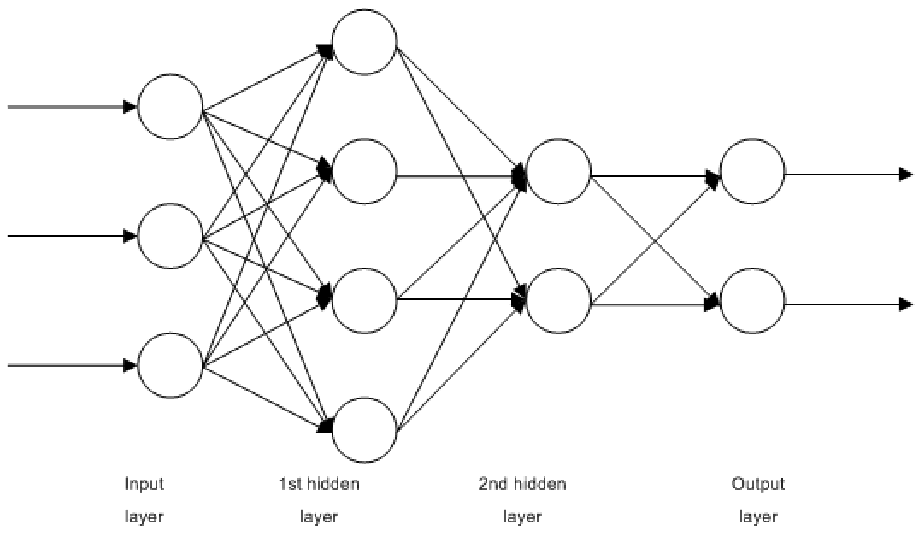


Figure 2.2: An Example of MLP

Backpropagation is a common way method to train a deep neural networks. The error of a neuronal layer is set to

$$f(w;x) = \sum_{k=1}^{K}(z_k(x) - z_k^*(x))^2 = \sum_{k=1}^{K}(z_k - z_k^*)^2 \tag{2.2}$$

$$z_k(x) = a(w'x)$$

where z_k is output of the layer and z_k^* is target output for input x, w stands for the weight matrix of the layer. $a()$ is the activation function

$$a(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

$$\text{and } \frac{\partial a(x)}{\partial x} = a(x)(1 - a(x)) \tag{2.4}$$

Gradient method is applied to rectify the deviation:

$$w_{t+1} = w_{t+1} - \gamma \frac{\partial f(w;x)}{\partial w_j}$$

where γ is learning rate, and from 2.2 we have

$$\frac{\partial f(w;x)}{\partial w_j} = 2[z_k - z_k^*]z_k(1 - z_k)x_j = \delta x_j$$

where $\delta = 2[z_k - z_k^*] \cdot z_k \cdot (1 - z_k)$ is error of neuron j of the layer.
The error of inner layer is determined by errors of all all neurons of subsequent layer and weights of associated connections. To have the error of hidden layers, firstly the error of output neurons is determined, then using these error to calculate the error of the preceding

layer and so forth until reaching the first inner layer. Let o_j be output of neuron j in layer S_m, so in general the backpropagated error of neuron j is

$$\delta_j = o_j \cdot (1 - o_j) \cdot \sum_{k \in S_{m+1}} \delta \cdot w_{jk}$$

2.4.2 LSTM

The LSTM (Long-Short-Term Memory)is first introduced by [11]. The LSTM unit can store its own information, which can be read, erased, and modified or independent of outside recurrent network and keep unchanged by additional input and output gate. The LSTM unit structure is shown as following 2.3. This unit is called *memory cell* and denoted c. Unit c current net input net_c is specified as

$$net_i(t) = \sum_j W_{ij} y^j (t-1),$$

where Wij is the weight from unit j to i at the last time step. In the following recurrent self-connection inclusive cell citing from [11] it has when u as other memory cells' output units

$$net_{out}(t) = \sum_u w_{out,u} y^u (t-1),$$

input units

$$net_{in}(t) = \sum_u w_{in,u} y^u (t-1),$$

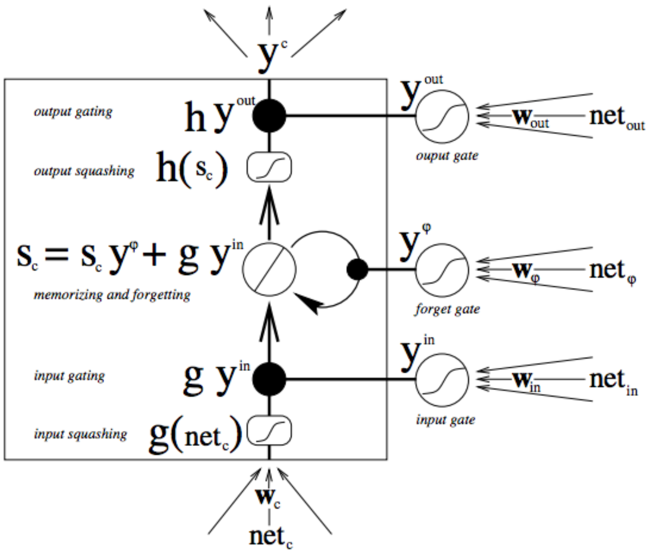
the forget gate units

$$net_{\varphi}(t) = \sum_u w_{\varphi,u} y^u (t-1),$$

memory units

$$net_c(t) = \sum_u w_{c,u} y^u (t-1).$$

When initial state where $t = 0$ the cell memory has "internal state" $s_c = 0$, which in later steps update itself by taking one part of input unit, squashed by differentiable function g, and another part of its memory with forget unit.



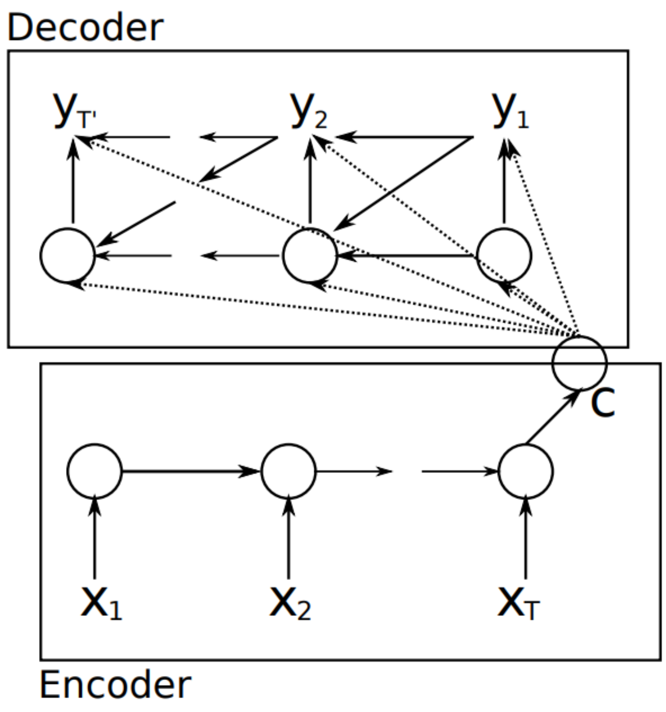
Source: `https://deeplearning4j.org/lstm.html`

Figure 2.3: Architechure of memory cell

In the later work the LSTM model will be constructed by using Tensorflow. The memory of LSTM unit called hidden state which include input, output, forget unit. The cross talk among units via weight matrix as the size of hidden state dimensionality which indicates diverse size of cell configuration is also needed as hyper-parameters of LSTM unit.

2.4.3 GRU

In paper of K.Cho [4] proposed GRU (Gated Recurrent Units) as the basic unit of RNN had also very good performance on sequence to sequence data processing like speech modeling and machine translation. As processing with small set of data GRU RNN is usually easier to be constructed(without output gate) and shown better performance than LSTM [5].
The encoder-decoder RNN has the following architecture. The bottom layer is encoder, which takes a sequence of data as input and "encode" the variable-length input sequentially. The hidden state C stores the summary of whole sequence input as a fixed-length vector representation. The layer above "decode" the fixed-length vector to generate variable-length of target output, in which both output y_T and hidden state h_T depend on y_{T-1}, as shown in 2.4. The input variable and target are no need to be in the same dimensionality, make out the utility of learning that the conditional distribution of a variable-length sequence is conditioned on another variable-length sequence.



Source:`www.machinelearningmastery.com/how-does-attention-work-in-encoder-decoder-recurrent-neural-networks/`

Figure 2.4: RNN with encoder and decoder

The 2.5 illustrates the hidden unit of GRU. The switcher z is update gate and r is reset gate, both are rely on input and hidden state, computed as following [4]:

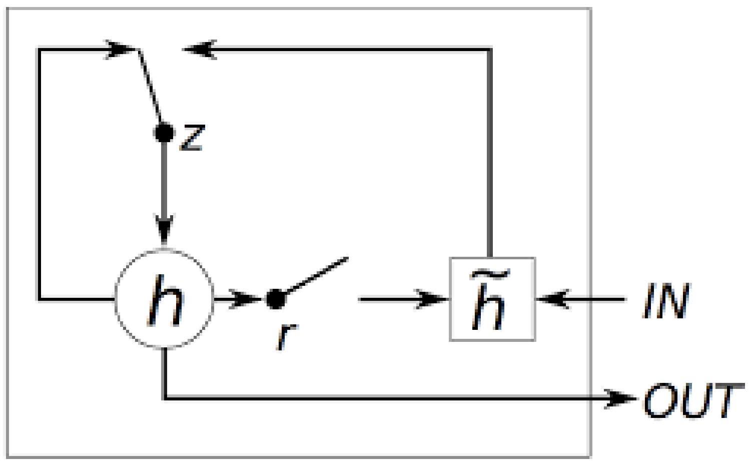
$$z_j = \sigma([\mathbf{W}_z\mathbf{x}]_j + [\mathbf{U}_z\mathbf{h}_{<t-1>}]_j)$$

$$r_j = \sigma([\mathbf{W}_r\mathbf{x}]_j + [\mathbf{U}_r\mathbf{h}_{<t-1>}]_j)$$

$[\ ]_j$ denotes the jth element of a vector, matrix U and W is obtained by training and h denote the hidden state and $\widetilde{h}$ is new hidden state:

$$h_j^{<t>} = z_j h_j^{<t-1>} + (1 - z_j)\widetilde{h}_j^{<t>}$$

$$\widetilde{h}_j^{<t>} = \phi([\mathbf{W}\mathbf{x}]_j + [\mathbf{U}(\mathbf{r} \odot \mathbf{h}_{<t-1>})]_j)$$



Source:`www.wildml.com/2015/10/recurrent-neural-network-tutorial/`

Figure 2.5: proposed hidden unit

Reset gate r decide how much the previous unit state is to be forgotten (abandon the previous unit when **r** is zero) and the update gate z control how much the hidden state h will be updated when new variables come in. The update gate is also considered as the memory mechanism of the cell, which helps in representing of short and long-term dependence. Apparently, the reset gate should be active if short-term dependence exists, also the update gate should be active for the account of long-term dependence.
In later work the encoder-decoder RNN with GRU model will be constructed for sequence data processing and sequentially prediction by configuring its network structure and GRU hidden state size.

Chapter 3

Model Selection

It is advisable to consider an ensemble as a combination of models with promising performance, then at least the weakness of a single model can be covered by the strength of others. But it does not mean that the more models are integrated, the better accuracy we will have. This is due to many reasons, e.g. the multicollinearity of forecasting models in an ensemble and because of the time-varying input data. Moreover, the optimal ensemble of a set of models also changes over time. In order to generate more accurate predictions, model selection is needed to find the optimal model or ensemble for varying time-series data. In this work, five candidates are to be selected by two model-selection approaches.

3.1 Models combination

To combine different models that complement each other or to strengthen their performance has been verified in many fields of science, because there is no such thing as a free launch, which means there is no single pattern or model that is suitable for all problems.An ensemble of neuron networks is a promising way to introduce new members in neuron network system[10] [10].

The five models proposed above have proven to be usable and to satisfy the basic requirement of prediction, but they perform differently. Therefore, it is reasonable to consider the question: How to make better progress with the individual models? In recent studies, building an ensemble is considered as a way for individual models to complement each other, assuming that they can take advantage of each other's strengths. There are multiple ways to combine the forecasting results from single models' outputs. Very common ensemble methods are bagging and boosting.Accuracy based K-nearest neighbors are used in [13] to combine different classifiers.

Dr. A. Chitra and S. Uma proposed a hybrid Pattern Prediction Ensemble in [3]. They used three non-linear models for time-series analysis. They combined these three prediction models using two-level ensembling, where first level concludes the heterogeneous

ensemble learning for different configuration of its member models, and in second level the best configuration is thus obtained and used in actual prediction. Through the two-level ensembling, the prediction error is decreased.
The most popular way to combine individual models is a weighted ensemble. In [18], a weighted ensemble construction is introduced that is based on prediction errors on sliding-window data. It proved that models can overall perform better in an ensemble. In our work, the ensemble is also built based on prediction errors.
Assuming that $M = \{M_1, M_2, ..., M_m\}$ is a list of models containing m models, in which every model is designed to model a given time-series, and that $V = \{V_{1t}, V_{2t}, ..., V_{mt}\}$ is the list of forecasting values to the next time step on the current time step t by M, the formula of the ensemble forecasting value for the next time step is as follows:

$$E_t = \sum_{i=1}^{m} \frac{F_{it} * (1 - \rho_{iH})}{\gamma}, \quad with\ \gamma = \sum_{i=1}^{m} (1 - \rho_{ciH}) \tag{3.1}$$

where E_t is the ensemble forecasting value using combination of individuals‘ forecasting value F_{it}, which are weighted by individual models' forecasting errors , ρ_{ciH} is model M_i‘s forecasting error on time window $[t - H, t]$, (see describing in section 4.2), H is the sliding window size, which indicates the error is aggregated from the current step to the last H steps, and H is the size of the sliding window and it is a user defined parameter. c indicates in which case the error is calculated. In our data there are only 64 steps for each case, models are trained case by case, and cases are independent of each other.

3.2 Dynamic ensemble members selection

As different testing data require different ensembles, selection of the dynamic ensemble members is needed. Through the predicted error outputs of the selected model, we are able to estimate the performance of candidate ensembles in the next time step. In a paper by Ko, Albert HR [13], four dynamic classic selection methods were applied. The most appropriate model was selected by its previous training data, which were similar to the test data, and in which previous predications were accurate. It is local accuracy-based model-selection.
In this thesis, an error-based model-selection approach will be used, and the model with the smallest error will be considered to generate the next prediction. In the paper [14], the model selection metric is the ensemble prediction error, which is predicted by building a learning model $E = f(X_i)$ that links the target value to the ensemble prediction error. Hence the possible error of the prediction can be calculated before the prediction itself. The following formula example is applied in selection procedure [14]:

$$E = \alpha + \beta_0 X_0 + \cdots + \beta_n X_n + \epsilon$$

where E is the prediction error, which uses root mean squared error(RMSE), α, β are regression coefficients, ϵ is the error term and X_i is the target value, where i is the index of time steps before the forecast time. These linear model parameters are selected using Ordinary Least Squares(OLS).
In [14], only one value of current input is used; using more than one value decreases the model's accuracy. In this work, it is also advisable to try new learning models like FNN (feed-forward neural network) and SVM (support vector machine), which have no linearity constraints and more advantages to describe the dependency between the forecasting error and target values. The forecasting error RMSE could be replaced with sMAPE, and the number of nearest target values in past is also to be optimised.
Different ensembles perform differently, which is not exclusively dependent on their component models. For example, model A performs better then model B, which is better then model C. However, it is still possible that the ensemble of A and C is better than the ensemble of A and B. Therefore, all individual models will first be randomly combined with all possibility, so that a set of candidate ensembles can be fairly constructed. Then, ensemble model selections will try to predict the error of each ensemble respectively, before using them for time series forecasting. The ensemble model with the best performance is selected to make the forecast.

3.3 Meta learning for models selection

The other approach to select individual models in steps is meta-learning. Meta-learning approaches are designed to select the best candidate model by extracting features of observations. In [16] commonly used time-series features like dependence, stationarity, and differencing are considered necessary meta-features. In [20], these features were applied as meta-features fed to a meta-learning model. The encouraging results in [20] have shown predictions on Time-Series Data Library (TSDL), and two meta-learning approaches were applied to select models for time-series forecasting, one used a single model selection algorithm – J48 decision tree – to select among two models. Another was to rank three models, the ranking was generated by sorting the scores, which were outputs of classifiers, implemented using multi-layer perceptron neural networks (MLP) associated with the models.
A meta-learning approach will be used for the model selection. Through a meta-learner, advisable models will be selected with diverse meta-features. Statistic features, model-based features as well as error-based features will be used as model input.
The meta-learning approach in our work has a similar structure to normal meta-learning, but it uses three different types of meta-features. Besides the statistic features of raw time-series as mentioned in [20] , the meta-features of de-trended and de-seasonalised time-series such as those proposed by Wang et al. in [23], and the error-based meta-features as Mirko et al. in [15] proposed are included in our work. We applied MLP for meta-learner con-

struction like Prudencio in [20] and Kueck in [15] did, the structure of MLP is given in chapter 4.
Meta-learning is a model selection method based on previously defined meta-feature sets and selects the optimal model according to its current input meta-features. The model selection procedure is executed by a so called meta-learner; in [15] the meta-leaner is built as FNN. The time-series features are extracted on the given training dataset, and combined with the best forecasting model on the same dataset to build a knowledge base to train the meta-learner (see the following figure 3.1).

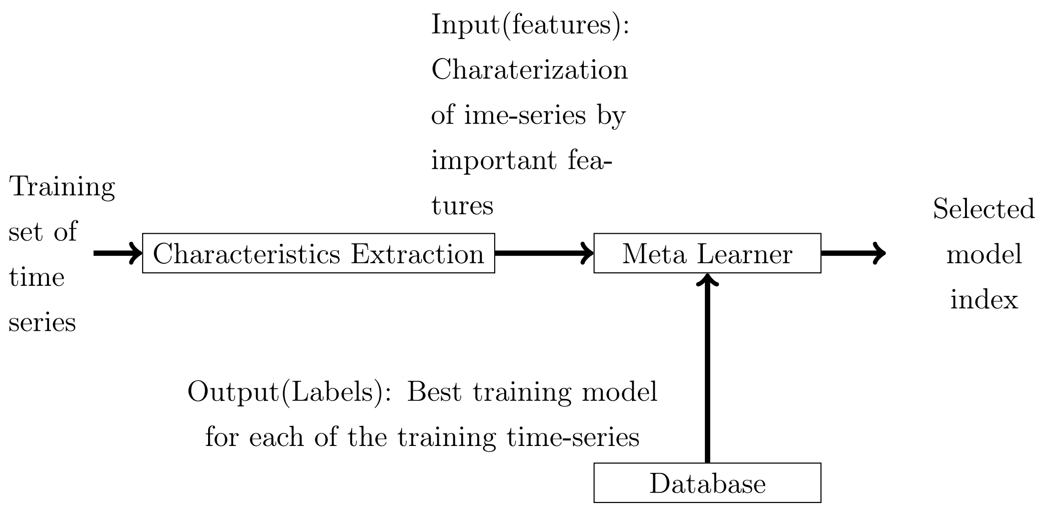

Input(features): Charaterization of ime-series by important features
Training set of time series
Characteristics Extraction
Meta Learner
Selected model index
Output(Labels): Best training model for each of the training time-series
Database

Figure 3.1: Training and deployment of meta-learner

Characteristics (meta-features) of an interval of a time-serie are computed and stored as the input features of the meta-learner. In training phase, the errors of all candidate models at each time step are sorted in ascending order, and the indexes of the sorted errors is stored as rank of performance, which are used to represent the output label of meta-leaner. Thus, the inputs and outputs are combined for meta-learner training by supervised learning. After the training, the meta-learner can be deployed for forecasting model selection with new input meta-features. Ensemble construction is also possible using meta-learning by attributing a ranking for the candidate models, instead of selecting one best model.
As for meta-features, three different approaches will be used, statistical or information-theoretic features, time-series orientated features, and error-based mate-features (landmarks). The three different approaches are to characterise our data. The statistic features measured on the raw time-series data. The statistic approach was used in [23] to characterise the raw data, in which 13 metrics were introduced. There are trend, seasonality, similarity, chaos, periodicity, serial correlation, skewness, and kurtosis. All implementation details are described in chapter 4.1.4.

Trend and Seasonality : Trend and seasonality are two main features to describe how time-series data evolve in general. The feature trend exists in long-term time-series. Trend indicates the changing direction of time-series, from an increasing to a decreasing trend. A seasonal pattern exists when a series is influenced by seasonal factors (e.g. the quarter of the year, the month, or day of the week). Seasonality is always of a fixed and known period.

Skewness and Kurtosis : The skewness and kurtosis are higher-order statistical attributes of a time-series. The skewness indicates a measurement of the symmetry of a distribution or a data set around its middle value. The normal distribution is symmetric and has a skewness of 0. If the number of samples with large value in a dataset is more than the number of samples with small value, this dataset is negatively skewed (left tail). Otherwise, if a dataset has more samples with small value than the number of samples with large value, then it is positive skewed (right tail), and has a positive skewness value. Kurtosis indicates the peakedness or flatness of a dataset. A kurtosis value close to three indicates a normal distribution like peakedness. Datasets with a relatively flat distribution have a kurtosis value of less than three; those with a sharper distribution have a kurtosis greater than three.

Similarity : In the area of classification, similarity is often regarded as a standard metric, which is subjective and highly dependent on the domain. A very popular similarity example is Euclidean distance. In our work, we use dynamic time warping to find the optimal non-linear alignment between two time-series. The Euclidean distances between alignments are then much less susceptible to pessimistic similarity measurements due to distortion in the time axis. We take the distance between the new incoming data and the previous best predicted data as a meta-feature.

Periodicity : A time-series is periodic if it repeats itself at equally spaced intervals. Since the periodicity of a time-series is important to find out its seasonality and cyclic pattern, it becomes necessary to have this value in meta-features. Fortunately, the time-series available in our domain come with known regular periodicity, 64 steps in each scenario. We take the frequency as a meta-feature based on the number of used steps.

Serial Correlation : Serial correlation describes the relation between a given variable and a lagged version of itself over various time intervals. Serial correlations are often found in repeating patterns, when the level of a variable affects its future level. We can calculate the correlation for time-series observations with observations over several time intervals. Because the correlation of the time-series observations is calculated with values of the same series at previous times, this is called a serial correlation, or an auto-correlation. The correlation coefficient in our experiment is for the varying point settlements, which are generated by random pressure and case-to-case independent, thus the relationship among them must be specified by the lag, Which should be within the same case or simulation scenario.

chaos : Chaos has been studied for a long time. It indicates the unpredictability and randomness of a non-linear dynamic system. Chaos is characterised by sensitive dependence on initial values, or more precisely by a positive Lyapunov Exponent (LE) [23]. In order to determine the random behaviour behind our time-series and its influence on our prediction process, it is necessary to take chaos into consideration. The LE of a dynamic system characterises the rate of separation (divergence) of two infinitesimally close trajectories. In our work chaos is quantified using LE.

Statistic feature set : In statistics, the feature set is used to describe the basic information of input data. Several statistics tests are also included. Our feature set of this part is similar to statistic features set as [15], which are listed in following: length of the time-series, minimum, maximum, mean, median, lower quartile, upper quartile, standard deviation, variance, coefficient of variation, linear coefficient trend test statistic, Mann-Kendall rank correlation coefficient, Spearman's Rho test statistic, Kruskal-Wallis test statistic, and F-test statistic.

Landmarkers : The difference between model-based meta-features and landmark meta-features is that landmarks do not stand for model characters but model performance. Landmarks describe the performance using the result of forecasts in the previous phase. Thus, models are used as labels in the training of a meta-learner. In our work, errors of models are used as meta-features to forecast a new dataset.
Error-based mate-features will apply the rolling-origin symmetric mean absolute percentage error (RO-sMAPE) in the following form to calculate the features.

$$\text{RO-}sMAPE^h_{i,i+I-1}(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{I} \sum_{k=i}^{i+I-1} sMAPE^h_k(\mathbf{y}, \widehat{\mathbf{y}})$$

$$\text{where } sMAPE^h_i(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{h} \sum_{t=i+1}^{i+h} \frac{|y_t - \widehat{y}_t|}{|y_t + \widehat{y}_t| * C}$$

$sMAPE$ is a scale-independent error and not biased to forecasting or an actual value. Actual y and prediction $\widehat{y}$ correspond with the input of window h from the original time step i, the constant C is manually adjustable, here set default value as 0.5. The sMAPE calculates the error from a fixed start point i, and aggregate itself in next I time interval. In this way the forecasting error, which can be a possible peak or valley by occurrences, when the prediction is coincidental to this origin, can be relatively more fault tolerant.
Therefore, the number of multiple forecasting origins I is set to average their error. The performance criterion of RO-sMAPE is considered as the first landmarks, the Training Error (TraE) of each forecasting model is also considerd as second landmarker. Training error binary ranking (TBR), which is indexes of a sorted list of binary meta-features, is also included in landmarks. TBR indicates the lowest, second lowest and highest error for all forecasting models (shown in the following section of forecasting models).
For the forecasting algorithms, the five models mentioned above (ARIMA, VARX, LSTM, GRU, MLP) will be used into the construction of the ensemble. Their results are extracted as landmarks.

Chapter 4

Experimental Results

4.1 Experiments

Our automated model selection framework contains individual models, ensembles, selection model, and a meta-learner. They work separately but also dependents upon each other and form a synthesis with multiple uses. Main uses include prediction, selection and self evolving. Predictions are accomplished by using different individual models of diverse types and their ensembles. As for the selection part, we chose two approaches described above. To sum up, the first selection approach is based on the direct prediction according to the raw data, without data pre-processing and direct feed to the neural network. The self-learning (backpropagation) of the neural network needs to find the hidden regularity of the raw data. The second selection approach is based on another view, the meta-learner, which takes meta-features as input. In addition to that, meta-features can be extracted through the processing to raw data. Meta-features represent the characteristics of data and so quantify the features of data in the form of real numbers. The extracted meta-features are not only an input, but can also be used to describe data and the performance of the forecasting models. For example, the TBR is the rank of performances of different models on previous data. The selection strategy for meta-features also influences the volume of the input. We used the meta-learner for individual model selection and the error-based selection model for ensemble selection. The model that is selected before the processing of the current step is changed, based on the output of the selection model and the meta-learner.

4.1.1 Hardware environment

Our work was developed and ran on a computer with an AMD Dual-Core A107850K processor, DDR3 RAM, and no accelerating using GPU. The operating system of the computer is Ubuntu 16.04 desktop.

4.1.2 Software environment

All work was accomplished with python. The statistic model was constructed with the library "statsmodels", the parameters of the ARIMA model were obtained with the help of another package called "pyramid", the LSTM, GRU, FNN and ensemble selection models and meta-learner were constructed with the libraries "tensorflow" and "numpy".

4.1.3 Data pre-processing

The data were divided into 3 parts, 60% for the training phase, 20% for the evaluation phase, and 20% for the test phase. Considering the structure of our raw data, the pre-processing work was necessary to fulfill the different input data demand.
The exogenous data, grouting and supporting pressures, were randomly machine generated in the range from 10,000 to 22,000. Compared to endogenous data, the point settlements in the range from -1 to 1, the exogenous data in that range were not suitable to be a part of input of our models. In our ensemble, we normalised them in the range of 0 to 1.
The error in each phase as well as the historical errors served as input for the selection model,the landmark of meta-features is the input of the meta-learner. For that, as well as for phase independence, all of the errors must be saved after the end of each phase. Because all parameters of models are randomly generated at the beginning of training phase, so there would show a randomly performance and the errors are unstable, especially for the neural network model. On this account, we took only the last 20% of errors from the training phase into account.

4.1.4 Experimental set-up

Individual model set-up
All models' setup parameters are set in the central control file, *config.json*. More details about the framework configuration are found in the results. We built the LSTM model with the code implementation of tensorflow BasicLSTMCell, the implementation is based on [25]. In order to reduce the scale of the forget gate in the beginning, the bias added to the forget gate is set to 1. The used LSTM model has a structure of four layers, each layer has cell units and hidden units sized $(128, 128), (154, 154), (154, 154), (154, 154)$. One layer of LSTM is constructed with parameters of dimensionalities of hidden state and output state. They must be equal. We have 154 outputs, so we chose a structure with softmax layers in size of 154. We have two hidden layers, which is enough for our model. More layers may bring better result but also bring the calculation with more time consuming and heavier model. We have hidden layers in the size of 154, instead of lighter, because there are 154 points. Points may be clearly identified even inside of the model.
With a similar structure to LSTM, we built our GRU model with four layers; the unit size

was also $128, 154, 154, 154$.
For the model MLP, we used a structure of five layers sized $156, 64, 32, 8$, and 154. The MLP was one more layer than the GRU and LSTM because MLP does not contain the memory unit, using one extra layer to redress the different. Using the same hidden layer size with LSTM and GRU may bring the MLP overflow effect in our work, so the smaller size of hidden layer was applied. The activation method was tanh. None of the three neuron networks took any bias due to the tiny change at each step. All neural networks were optimised in the training phase using the Adam Optimizer [12] with a learning rate $0.005, 0.001, 0.00005$ respectively.
It was actually tough to determinate the order (p, d, q) of ARIMA model. The automatic ARIMA order selection we implemented took much longer computing time then ARIMA itself, and did also not give us a promising result. After times of applying pyramid, we determined that the order $(3, 0, 0)$ was most suitable for our work. For the VARX model, the maximum number of lags to check for order selection was set to eight.

Ensembles set-up
In our experiment, we proposed 10 ensembles, covering all possible combinations of individual models. ${C_3}^5 = 10$ ensembles were built for dynamic ensemble selection. The input of the ensembles include the forecasting result from the individual models. The ensemble took also the error aggregated from the previous steps, which was in size of customised window of the input time-series. Therefore, the number of steps of the output of ensembles are less than as the of individuals. In our experiment, the steps of ensemble output in the evaluation and test phase were:

$$ensemble\_output\_steps = ensemble\_outputs\_steps - error\_window\_size$$

and had the number as $1891 = 32 * 61 - 61$.

Selection model et-up
As explained above, the selection model was used for ensembles. For each ensemble a selection model was trained. In sum, 10 selection models were trained and saved for further use.
In each case, a selection model takes endogenous variables in the first sliding window also the exogenous variables in al 64 time steps as inputs. The output is a real number, which indicate the mean error in this case, that the model may produce. Therefore, we can estimate a model's error through its selection model. The selection models for ensembles can only first be trained in the evaluation phase, because the ensemble input needs the error data from the last phase. In order to separate different phases, the errors from each phase were saved. This was practical as the errors were calculated every 64 steps in our

experiment, thus the saved data were much smaller than the raw data, and we did not have to train or evaluate the old data every time when we built an ensemble or selection model.

Considering model saving and implementation, the selection models for individual models and ensembles were built according to the same structure, namely a Multilayer Perceptron (Neural Network) with layers sized $64, 32, 8, 1$. The input of the selection model in this experiment includes all exogenous pressures in current cases and also the settlements at the first sliding window, which are given for further prediction of the case. Output is a real number, which indicates the average error of all points for one case. It is also optimised by the Adam optimizer with the learning rate 0.01.

Meta-features set-up

For the extraction of meta-features we first concatenated the errors, settlements and pressure for each candidate. Meta-features such as trend and seasonality as well as chaos etc. were extracted from a sequence of continuous historical data, which should end until the actual time step, moreover the size of the sequence should equal as the size of a case, i.e. 64 time steps. Meta-features were extracted separately for each point. Every training and every evaluation of the meta-learner were orientated on one case. The meta-features were extracted from 16 cases, which had just been completed. Model meta-features were extracted from point settlements to characterise the time-series, and statistical meta-features were extracted from the exogenous pressures.

Meta-features were extracted using methods from different python modules. Trend and seasonality were calculated using seasonal_decompose from "statsmodels tsa seasonal" with parameters additive seasonal component, frequency 64. From the "scipy stats" model we imported the method for skewness and kurtosis.

As serial correlation, we calculated the auto-correlation of our time-series by using the function "acf" from "statsmodels.tsa.stattools", which does a full convolution. We set the number of lags as the number of cases for the return value of the functions.

For similarity, we chose the Euclidean distances between the current exogenous time-series to the exogenous time-series used in the previous prediction, i.e. we differentiated the previous predictions by comparing them with the case in the current step. We used previous predictions in the same size as TBR, which was one in our work. The function was "norm" provided by python module "numpy.linalg".

Nolds is a small numpy-based library that provides an implementation and a learning resource for non-linear measures for dynamical systems based on one-dimensional time-series. Positive Lyapunov exponents indicate chaos and unpredictability. Nolds provides the algorithm of "Rosenstein et al.". We used "$lyap\_r$" to estimate the largest Lyapunov exponent to indicate the chaos of our time-series.

For the statistic features, the "numpy" and "scipy" met all of our needs. The exogenous pressures have direct influence on settlements, and could be easily characterised by statistic features in the above chapter 3.3. As for landmarks we took the ro-sMAPE as TraE (training errors) for all points over the last 16 cases. For TBR, we first calculated the rank of each individual model according to their TraE, based on which we could calculate the TBR in the formula:

$$TBR = 2^{-R_i}$$

where R_i is the in0dex of the rank of model i, e.g. if LSTM is in a time step has performance as third good in all candidates, so R_{LSTM} is 3 and $TBR_{LSTM} = 2^{-3} = 0.125$.

Set-up of meta-learner for individuals

The meta-features were calculated from the last 61 steps and reshaped as a one-dimensional array, as the input of the meta-learner. Because the meta-features of points are irrelevant to each other, we chose a relatively larger FNN with layers sized $180, 160, 154, 64, 32, 16, 5, 5$. The output of the meta-learner indicated the predicted rank for the five individual models, which described the performance in binary form as the current steps. We set the learning rate to 0.02 for the training process.

Set-up of meta-learner for ensembles

For better use of the existing ensembles, we also applied the meta-learning approach for the selection of the best ensemble, which actually proved also that meta-learning is a promising approach for model selection. All meta-features of the meta-learner for ensembles are same with for individuals stated in chapter 4.3.3. The neural network's structure and configuration parameters were also same, except the size of input and output because of number of ranked candidates. The meta-learner for ensembles has 10 candidates indicating 10 ensembles.

4.2 Evaluation metrics

When we fed error data to train the selection ensemble, and as meta-features of meta-learner, several error metrics were used. For comparability and unity, the same metrics were used as individual models' error, ensemble models' construction, and input of ensemble selection, as well as meta-features of the meta-learner. We used the following errors as measurements, and compared them in a table.

Symmetric mean absolute percentage error $sMAPE$ is an error measurement of a forecasting model based on a percentage error:

$$sMMAPE = \frac{100\%}{n} \sum_{t-1}^{n} \frac{|\hat{y} - y|}{(|\hat{y}| + |y|}) / 2$$

where A_t is the true observation and F_t is the forecast value.

Root mean square error (RSME) could also be an error measurement of the forecasting model between forecast values and actual observations. It shows how the forecast value deviates from the true value with the following formula:

$$RSME = \sqrt{E((\hat{y} - y)^2)}$$

where $\hat{y}$ is forecast target value and y is the true target value. The RSME is not quite adaptive in our experiment, because the value observation data is too small. We cannot see the changes well over time steps through tiny fluctuation of the RSME.

Normalized root mean square error (NRMS) normalise the RMSE changing the error in percentage form. In our experiment, we calculated the normalisation as follows:

$$NRSME = \frac{RSME}{y_{max} - ymin}$$

where y_{max} and $ymin$ are maximum and minimum value of target measure data variable. Using the difference of y_{max} and $ymin$ as the approach of normalisation is adaptive in our experiment, because our observation ascend continually over time steps, so y_{max} and $ymin$ are never equal.

4.3 Results

4.3.1 Individual models

The starting point of our work was to have accurate models. The models are well built and trained, and show a considerable performance. The following graphs shows the result of LSTM on point 143:

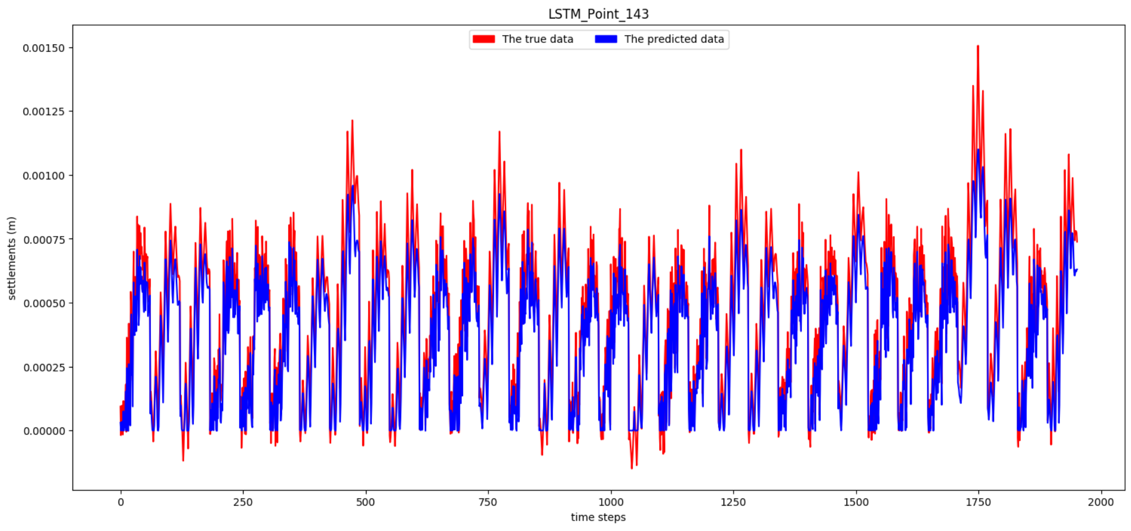


Figure 4.1: Predicted versus real settlements on point 143 using LSTM

The following graphs shows the result of GRU on point 143:

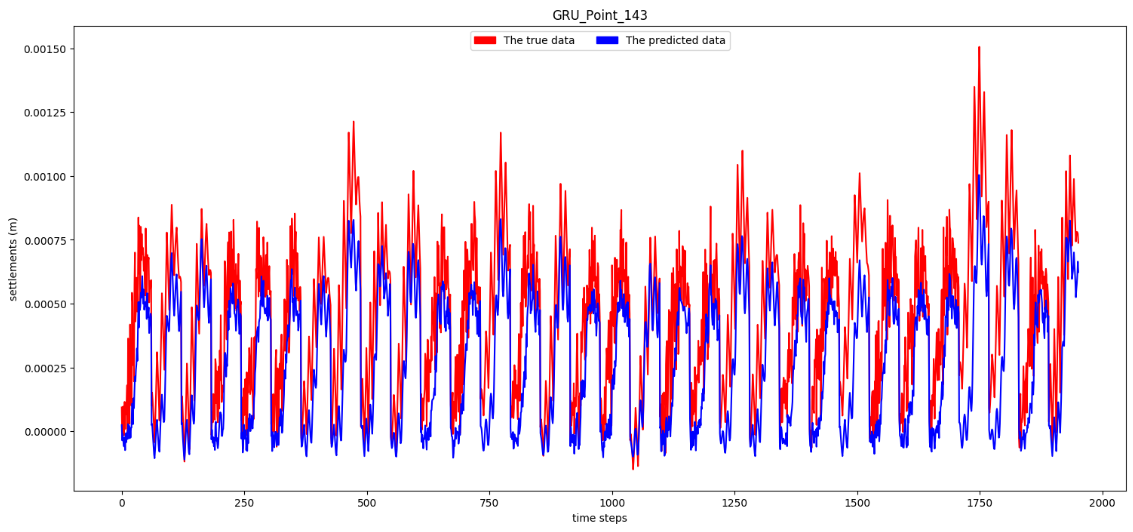


Figure 4.2: Predicted versus real settlements on point 143 using GRU

The following graphs shows the result of MLP on point 143:

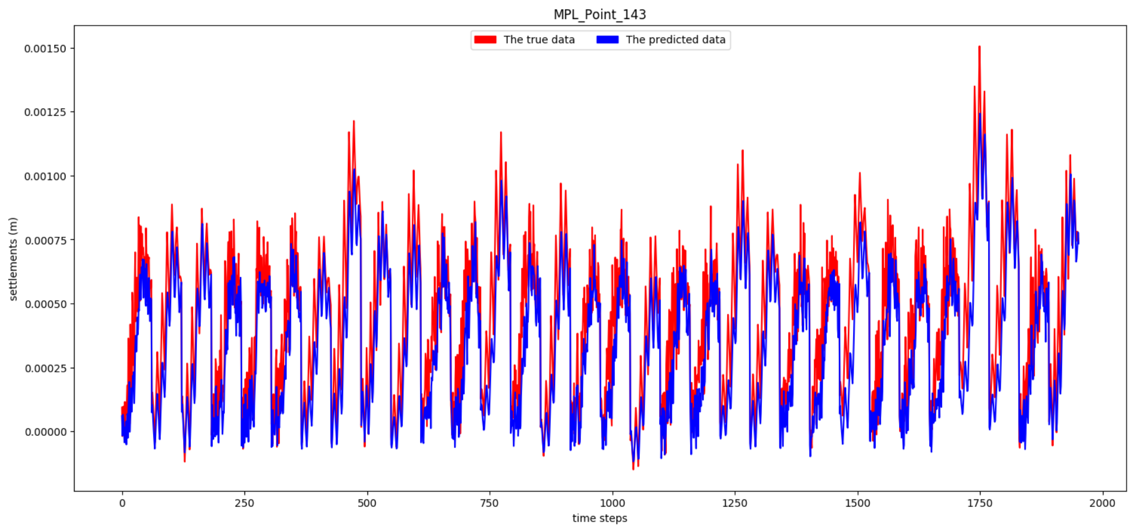


Figure 4.3: Predicted versus real settlements on point 143 using MLP

The following graphs shows the result of GRU on point 143:

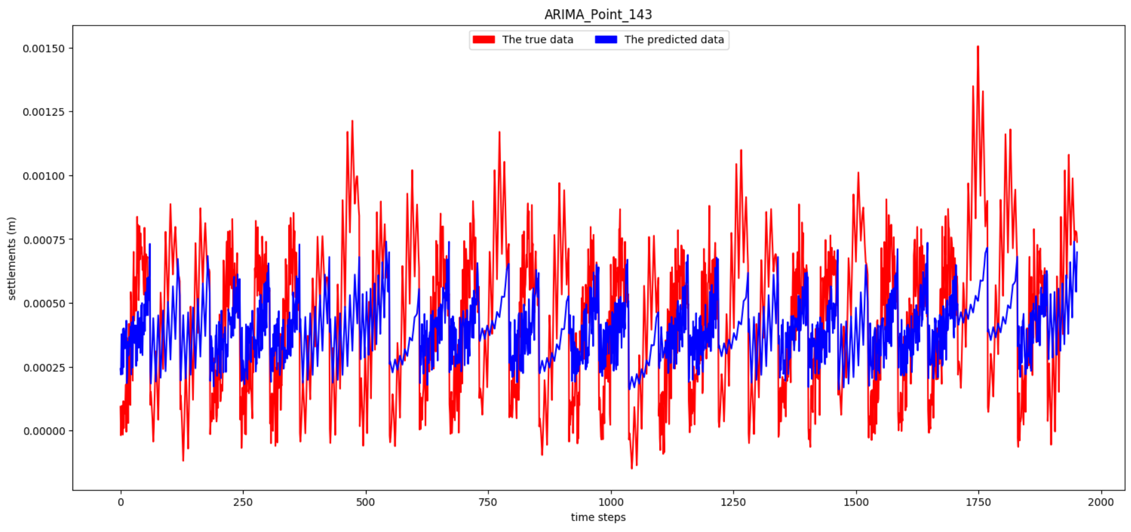


Figure 4.4: Predicted versus real settlements on point 143 using ARIMA

An important reason why we concatenate cases together is, that a considerable number of prior values are necessary both for ARIMA and VARX. The VARX and ARIMA both take all steps in training phase as prior values. When they make prediction in evaluation and test phase, the prediction is performed for a whole phase. Unlike neural network models, which can take three steps of prior value and predict a step at each time, the normal performance of ARIMA and VARX require more prior values for regression analysis. With several steps as prior values only invoke exceptions in our experiments.

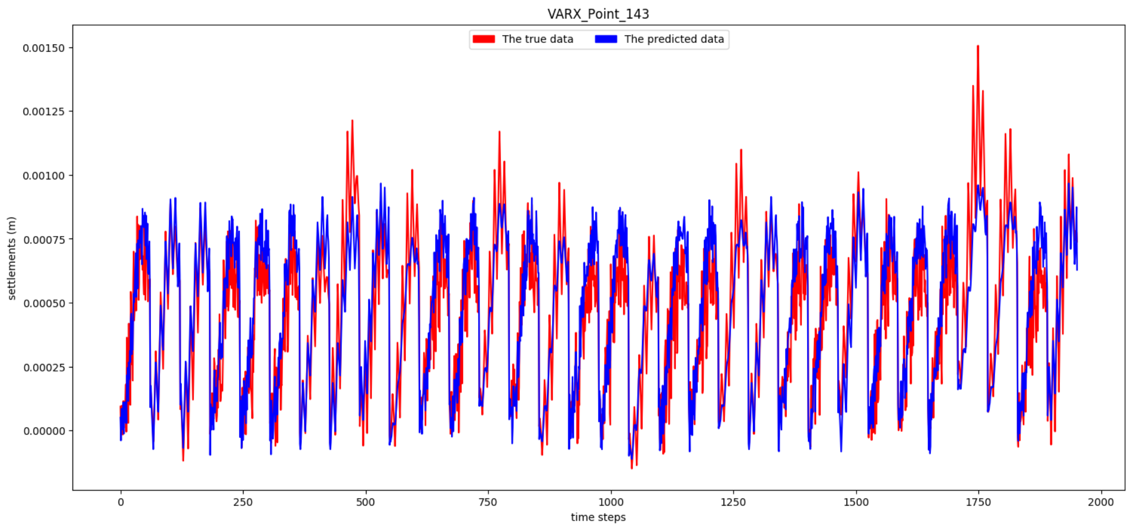


Figure 4.5: Predicted versus real settlements on point 143 using VARX

Here are some graphs showing the general performance of individual models. The error graph part is shown with models generated before the model selection approach.

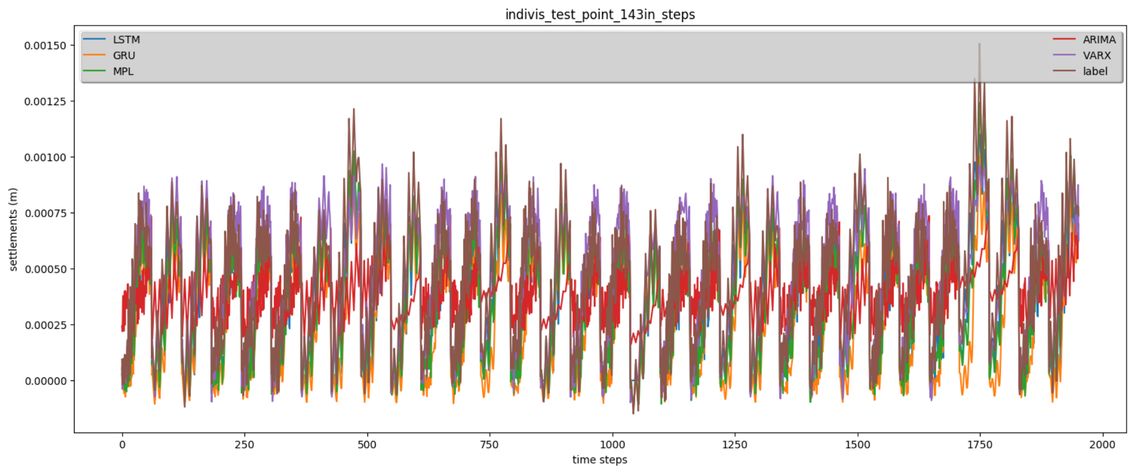


Figure 4.6: All individuals result in the test phase with the label of point 143

Based on the above and graphs in the Appendix A.0.1, it is clear that models perform differently on one point. The LSTM model and the VARX model show the comparatively best performance. In contrast, ARIMA is intuitively not good at this problem, and should therefore be avoided in the model selection. Compared to the LSTM and GRU recurrent neuron networks, the MLP has poorer performance. In the regression analysis field, the neuron with memory effect is reasonably more adaptive to time-series forecasting than MLP, which is usually suitable for classification problems.
In our experiment, because the period for raw data was relatively short the models required a large data volume,the result of actual step may be predicted using input from the last case. So, the performance of the neural network was not necessarily better than the statistical models.
Another weakness of the statistic model was the quantity of the input. Both ARIMA and VARX need input data with more than thousand steps to further forecasting. In our experiment, the data were actually separated into cases, and each of them were limited to only 64 steps. Therefore, we concatenated the cases for the use of the model. Moreover, the parameters of the statistical model, such as the order of ARIMA and lag of VARX, are more relevant for statistic expertise. The parameters we used in the above statistical model were not the most adaptive. As for general machine learning algorithms, the neural network has higher abstraction level and is more flexible for various problems. For more details of individual results, see the Appendix A.0.1.

4.3.2 Ensemble selection model

The selected enesemble with the smallest predicted error was considered to use for forecasting. The following graphs shows the performance of individual ensembles and the selected ensemble. Although there are 11 lines in the error plot, it is not difficult to see that the selected ensemble show no improvement among all candidates.

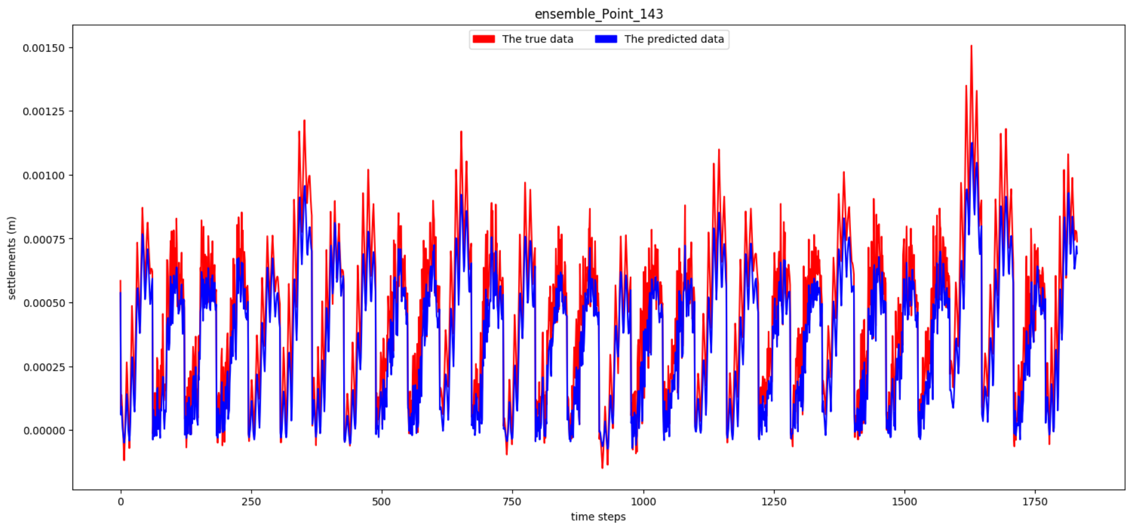


Figure 4.7: Predicted versus real settlements on point 143 using selected ensemble

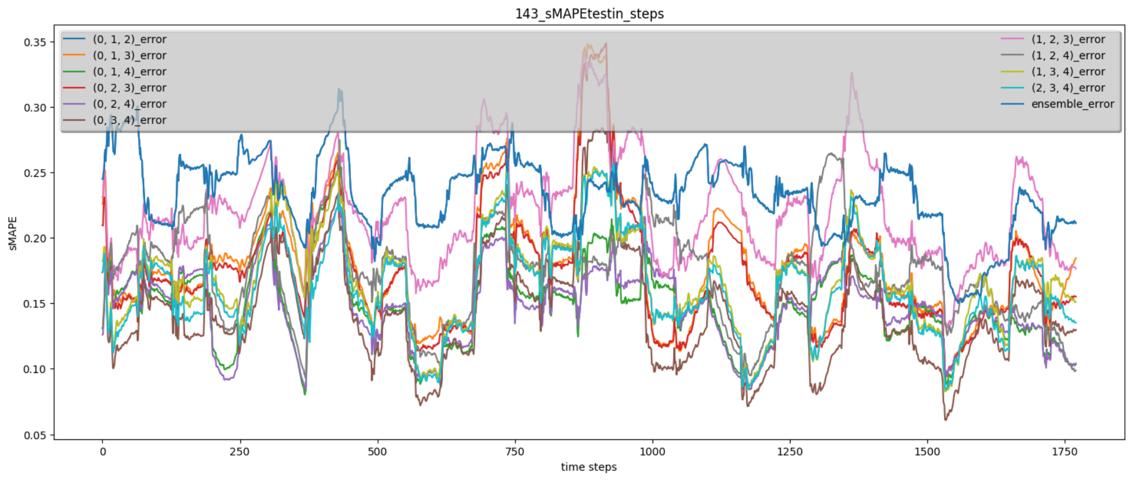


Figure 4.8: Selected ensemble error on the test phase on point 143

where the number 0,1,2,3,4 in the suffixes of ensembles indicate its member models, with the order of LSTM, GRU, MLP, ARIMA, VARX. We use the same indexing rule for later graphs with ensembles, also the table 1.2.1.

4.3.3 Meta-learner

The difference in structure between the error-based ensemble selection in 4.3.2 and the meta-learner is that the meta-learner predicts the ranks of five individual models, which is the output for 154 points respectively, not like the ensemble selection, which only has an output of one error. The reason is that meta-learner contains meta-features, which are independent for every point. Because the meta-learner must be trained and saved for every single point separately, we have to train 154 models for each matrix to the determine the actual performance of the meta-learner.

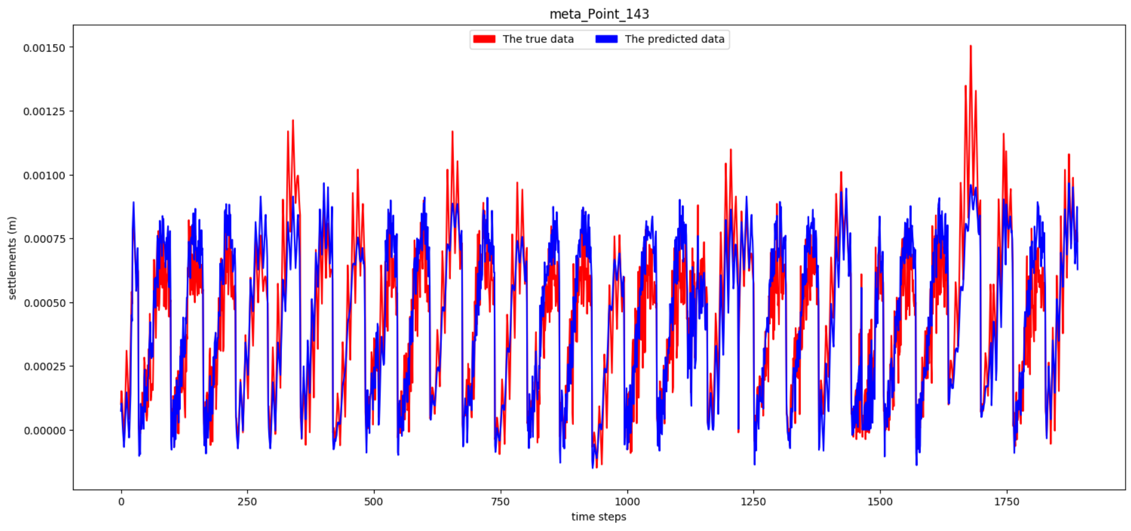


Figure 4.9: Meta-learner result in the test phase with the label of point 143

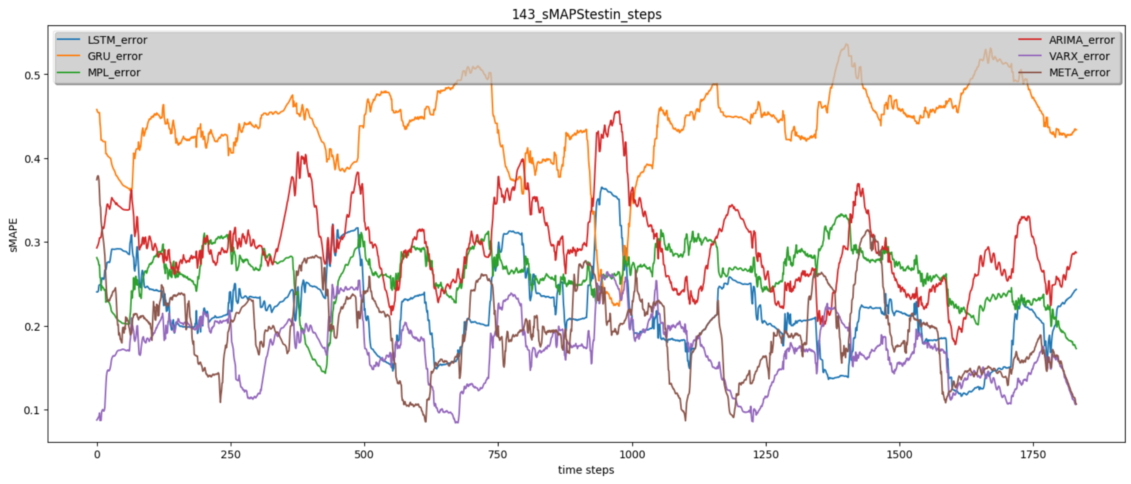


Figure 4.10: Selected individual's error in the test phase point 143

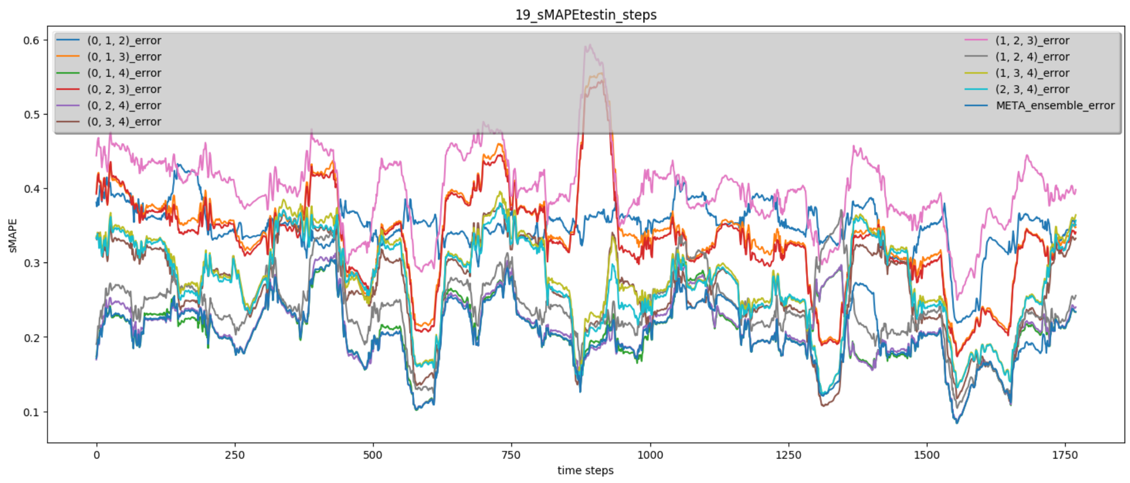


Figure 4.11: Selected ensemble's error in the test phase of point 19

For point 143, meta-learner is actually not absolutely better than the VARX model over all the time steps, but meta-learner still have the performance in general. Because of 160 points we had, there are many graphs, which show the evident improvement of meta-learner. We give some other graphs listed in the section Appendix. The performance of point 30 in figure A.7 A.10, point 38 in A.8 A.11 and point 47 in A.9 A.12 are listed in Appendix also both in result and error, and it could be found that the meta-learner has the best performance.

4.3.4 Average errors and meta-learner for ensembles

We used meta-features to represent the original data in the meta-learner. Such features are extracted in many aspects, so that the input to the meta-learner is distinctly enlarged. Therefore, we must train the meta-learner for point independently. For a complete prediction of meta-learner performance for all points, we must train $462 = 154*3$ models, because different error metrics as meta-features and outputs should be separated. The meta-learner using $sMAPE$ as meta-features may be unsuitable to select ensembles, which is calculated based on $NRSME$, vise versa. Following are error graphs for ensembles selection by meta-learner: figure 4.12 of point 23, figure 4.13 of point 115, figure 4.14 of point 143. For the same points, comparison graphs are also shown below (comparison graph is consisted by ensemble selection approach, meta-learner for individuals and ensembles):

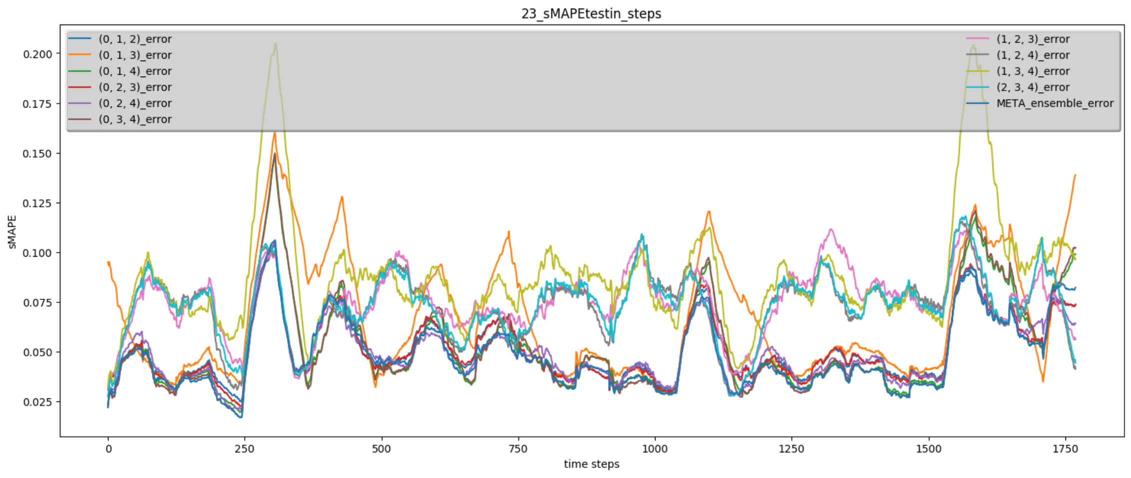


Figure 4.12: Selected ensemble with meta-learning error in the test phase of point 23

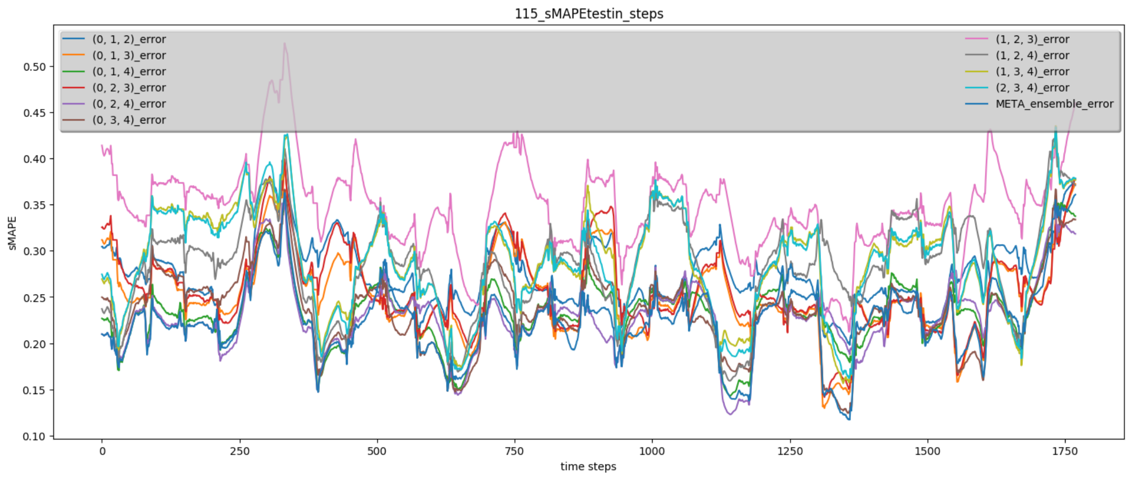


Figure 4.13: Selected ensemble with meta-learning error in the test phase of point 115

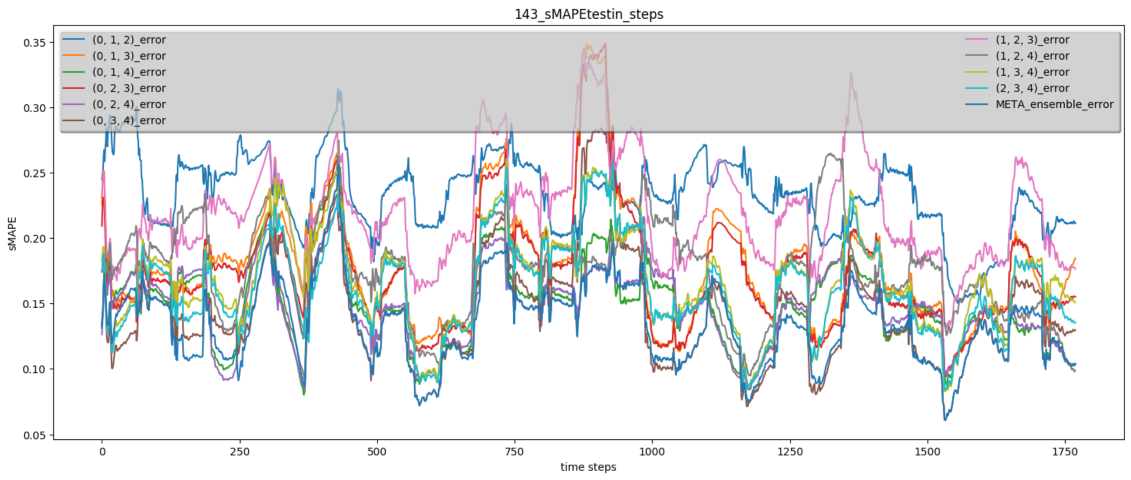


Figure 4.14: Selected ensemble with meta-learning error in the test phase of point 143

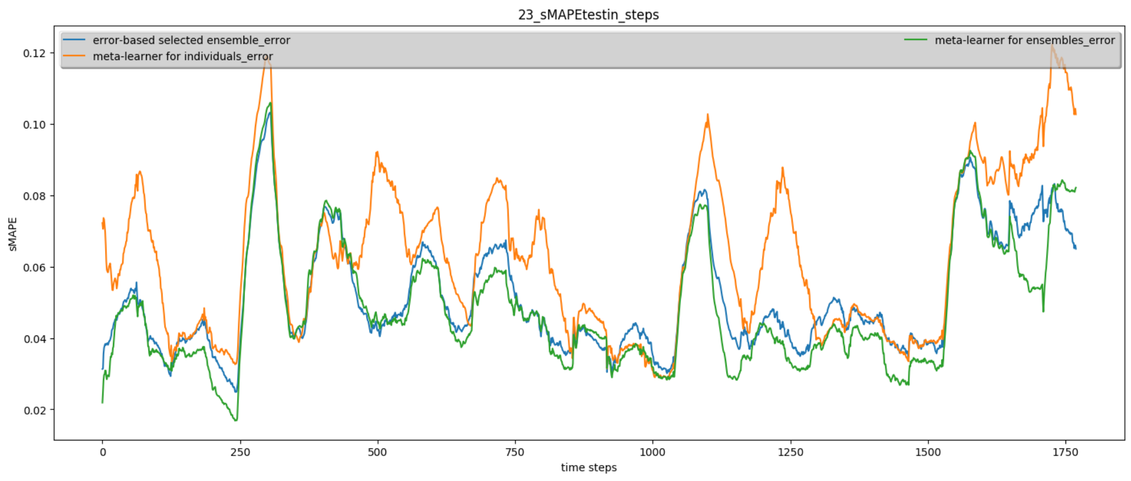


Figure 4.15: Errors of models selected by thee different approaches in the test phase of point 23

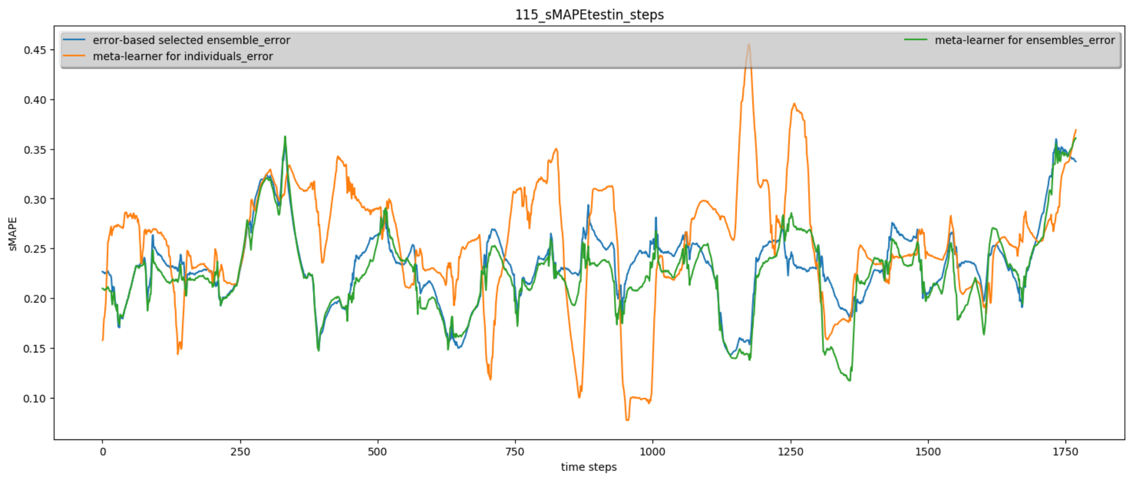


Figure 4.16: Errors of models selected by thee different approaches in the test phase of point 115

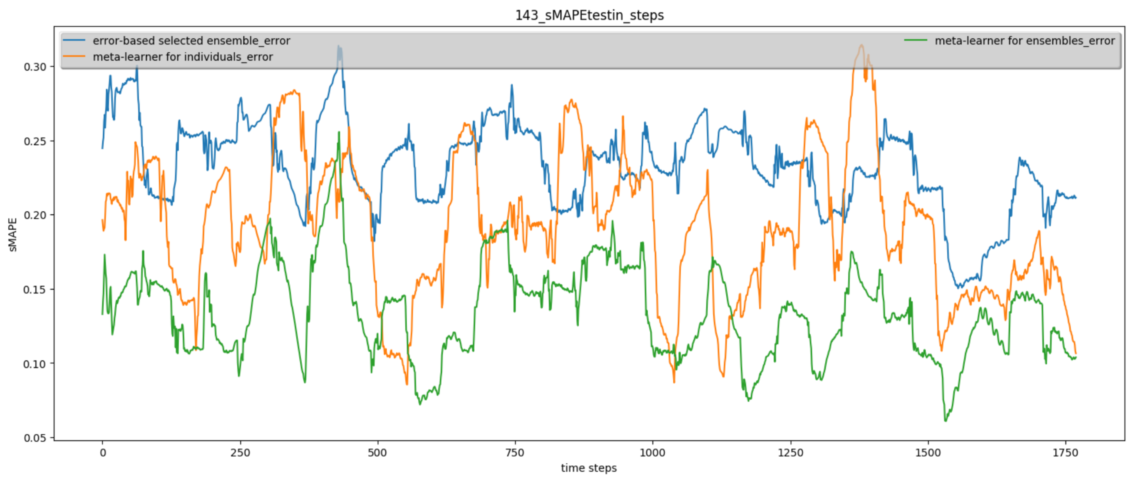


Figure 4.17: Errors of models selected by thee different approaches in the test phase of point 143

4.3.5 Average error table

The average of errors and the standard deviations over all points in the test phase are also shown in the following Table 4.1.

The error values in Table 4.1 are average errors over 154 points and over all time steps in

	sMAPE	**NRSME**
LSTM	16.24% ± 12.16%	4.80% ± 1.89%
GRU	30.02% ± 19.40%	11.81% ± 4.72%
MLP	23.71% ± 16.16%	9.42% ± 3.42%
ARIMA	39.81% ± 16.94%	18.08% ± 4.64%
VARX	15.81% ± 8.04%	7.07% ± 2.40%
ensemble(0, 1, 2)	16.47% ± 13.07%	4.96% ± 1.81%
ensemble(0, 1, 3)	15.55% ± 11.73%	5.35% ± 2.22%
ensemble(0, 1, 4)	12.26% ± 8.25%	4.45% ± 1.52%
ensemble(0, 2, 3)	15.04% ± 11.86%	4.90% ± 1.69%
ensemble(0, 2, 4)	12.18% ± 8.24%	4.22% ± 1.15%
ensemble(0, 3, 4)	12.63% ± 9.09%	4.52% ± 1.55%
ensemble(1, 2, 3)	22.07% ± 15.26%	9.25% ± 3.25%
ensemble(1, 2, 4)	14.78% ± 9.26%	5.88% ± 1.52%
ensemble(1, 3, 4)	15.52% ± 9.46%	6.54% ± 2.01%
ensemble(2, 3, 4)	15.04% ± 9.74%	5.99% ± 1.46%
Selected Ensemble	15.17% ± 11.75%	5.07% ± 2.13%
Meta leaner Individual	14.79% ± 9.54%	4.71% ± 1.71%
Meta learner Ensemble	11.37% ± 7.99%	4.09% ± 1.14%

Table 4.1: Average errors with metrics sMAPE and NRSME over all 154 points in test phase, where the number 0,1,2,3,4 in the suffixes of ensembles indicate its member models, with the order of LSTM, GRU, MLP, ARIMA, VARX.

the test phase. For all candidate and selected models, the average error results of sMAPE and NRSME are

$$value = \bar{e} \pm std(e)$$

where e is error over all time steps of actual phase, $\bar{e}$ is the average error and $std(e)$ is the standard deviation of error e.

The individual models are located in the first five rows. With metrics of sMAPE the VARX model had the best performance, with NRSME the LSTM model did. The MLP model had lower error results than the GRU, and the ARIMA model did not perform better than others in general.

The ensembles are named with the indexes of their members and located in first column

directly below individual models. The performances of ensembles were not necessarily better than those of their members. But for those ensembles that were not better than their best member, they still maintained the same level of performance as their best member. For example, the ensemble with the members LSTM, GRU, MLP had a sMAPE value of 16.47%, and was approximately equal to LSTM with a value of 16.24% and better than the GRU and MLP models. For the ensembles that included the VARX model,all had a sMAPE value lower than the VARX model.
The row behind the ensembles is the error-based selected ensemble over time steps. It shows literally no apparent improvement than its ten ensemble candidates, not better than the best candidate with sMAPE or NRSME. The last two rows shows the average errors of meta-learner for candidates of individuals and ensembles. Meta-learner for individuals has sMAPE of 14.49%, which is the lowest value among all its candidates. Meta-learner for ensembles has sMAPE 11.36%, which is also the best performance among all ensemble candidates. For the metrics NRSME, the meta-learners for the both kinds of candidates have the same performance as the meta-learner with metrics sMAPE does.

4.4 Discussion

The neuronal networks displayed not necessary better performance than the statistical models. The ARIMA had good performance only on some of specific points and VARX had also unstable performance on some of points, e.g. the average errors of five individuals for point 38 were listed in table 4.2

	LSTM	**GRU**	**MPL**	**ARIMA**	**VARX**
sMAPS	3.79%	6.92%	3.962%	17.70%	7.59%

Table 4.2: sMAPS of five individuals for point 38

But the three neuronal networks performed more stable when cases and points changed. This is due to the complexity of neuronal networks, which makes these models robust. Neuronal networks are able to adjust themselves over time and have comparatively more strengthen in detecting the non-linear dependency. On the other hand, the statistical models are set with parameters, which are constants, and make them limited at modelling over time. Accounting on the performance of VARX, linear dependencies between historical exogenous and endogenous variables were successfully modeled using statistical models and neuronal networks.
The ensembles with individual models generally showed a satisfactory performance. Although the improvement was not perceptible in some cases, the performance was stable and boosted the performance of its members in effect. The performance proved that the combination of members' results with the weighting method stated in Equation 3.1 is re-

liable in avoiding extreme cases and improving the accuracy.
The error-based model selection approach did not achieve a good result in our experiment. Trying to predict the ten errors using FNN is difficult to achieve, because the errors of its candidates are usually close. Neural networks are not able to detect such small differences. The error of a neural work itself cannot be tolerant and often makes the backpropagation correction undirected and chaotic. Using error as the target value of FNN makes the FNN training ineffective, and does not utilize its strength.
The meta-learner approach made the model selection robust, as shown in Table **??**. It evidently improved the accuracy for both kinds of candidates and both metrics. The meta-features as input provided more specific information comparing with the input of endogenous and exogenous variables. MLP use back-propagation to train itself, so it is target label value oriented, instead of trying to find out any features. Therefore, our meta-features may not be produced inside of neuronal networks, but they could be actually helpful for our experiments. So, although a similar structure of MLP was applied in the error-based model selection approach, the meta-learner solved the problem more effectively and efficiently.
Compared to the error-based model selection approach, the meta-learner used meta-features as input and classified the error values as its targets, which makes the meta-learner adjust itself well advised, when the best model among candidates switchs itself over time. The meta-learner does not always select the best performing candidate in Table **??**. More robustly, it adjusts itself over time, so it tries to select the best candidate at each time step,that is why it has a significant improvement over its candidates.

Chapter 5

Summary and Outlook

The applied individual models are very popular nowadays and have been shown to be successful in solving a wide range of problems. Mutated models are usually explicit to specific problems. Models like convolutional LSTM recurrent network and Grid LSTM recurrent network as well as vector support regression are also applicable in our experiments.

Our framework is designed at a low coupling degree. Hence, it is easy to update, e.g. by adding new individual models or selection approaches. All configuration details are set in a central control file in JSON format. When we run the whole process, all functional modules are combined together and processed sequentially. For the experiment, all modules are combined and processed synchronously. It could be time consuming, because not all of them are needed for actual use in practice. In our experiment, data are generated periodically. Therefore, very similar cases may apply a similar or same routine of model selection over time, through which only the required modules are processed, to accelerate the process of forecasting.
The meta-learner approach for model selection worked better than the error based approach. During the training we found many applied meta-features are constants over time steps, or only have value 0 or 1. Therefore, many applied meta-features could be excluded. Keeping the meta-features compact not only indicates a comprehensive understanding of source data's but also speeds up the training of the neural network. With the good performance of meta-learner in our work, we may apply meta-learner in more fields and meta-feature for more data types, such as images, voices.
Time-series analysis should be scalable and robust, because the dependency between exogenous and endogenous variable may change over time. The more features could be extracted along with deeper understanding of a time-series. That could also bring more perspectives, in that our work on time-series data could be a virtuous cycle.

Appendix A

More Information

A.0.1 Individual results

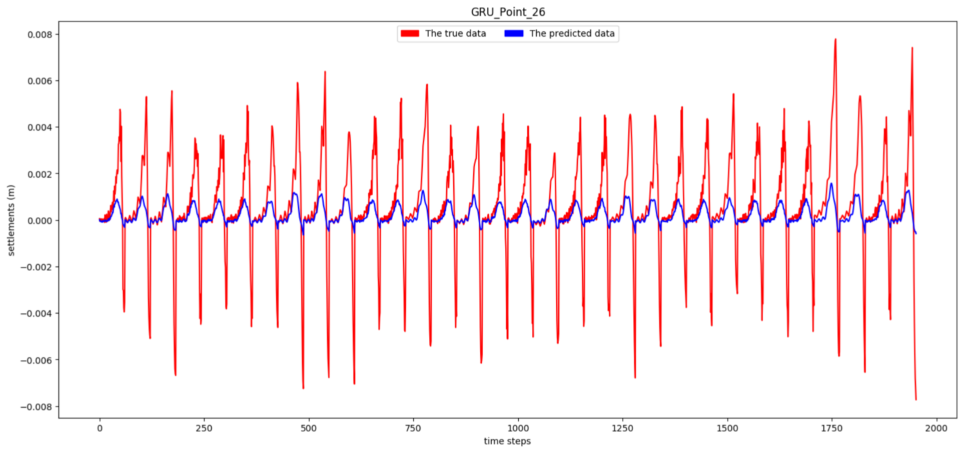


Figure A.1: GRU model result in test phase and the label of point 26

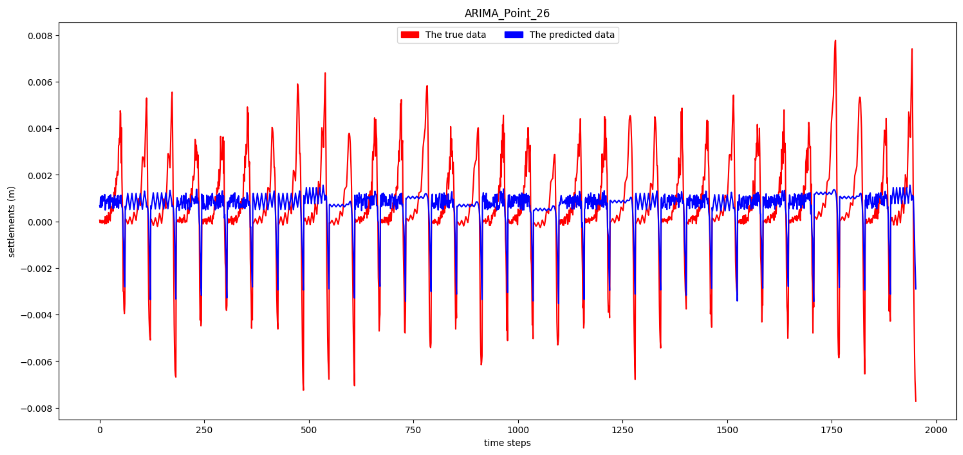


Figure A.2: ARIMA model result in test phase and the label of point 26

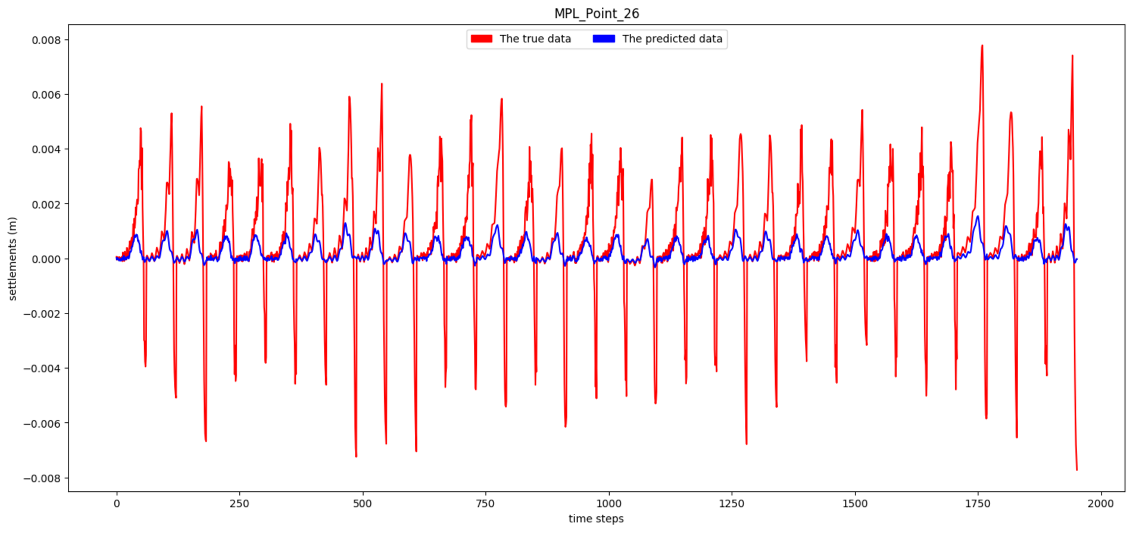


Figure A.3: GRU model result in test phase and the label of point 26

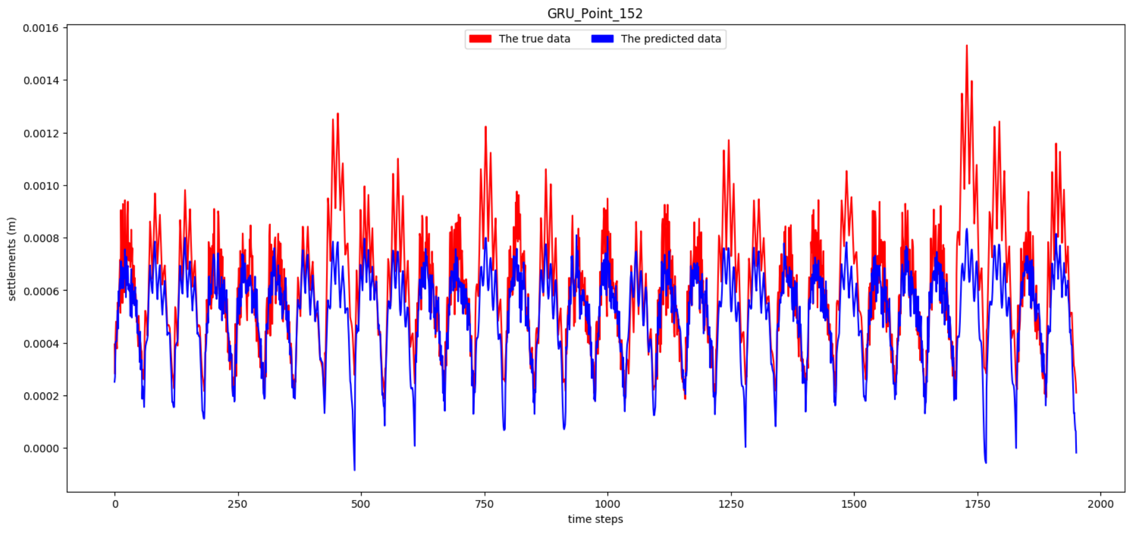


Figure A.4: GRU model result in test phase and the label of point 152

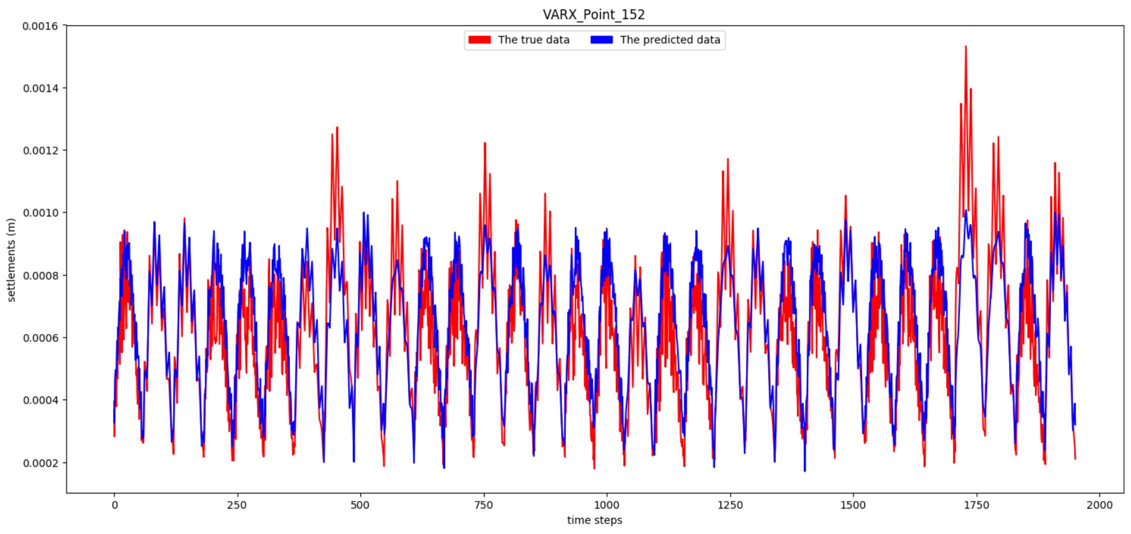


Figure A.5: VARX model result in test phase and the label of point 152

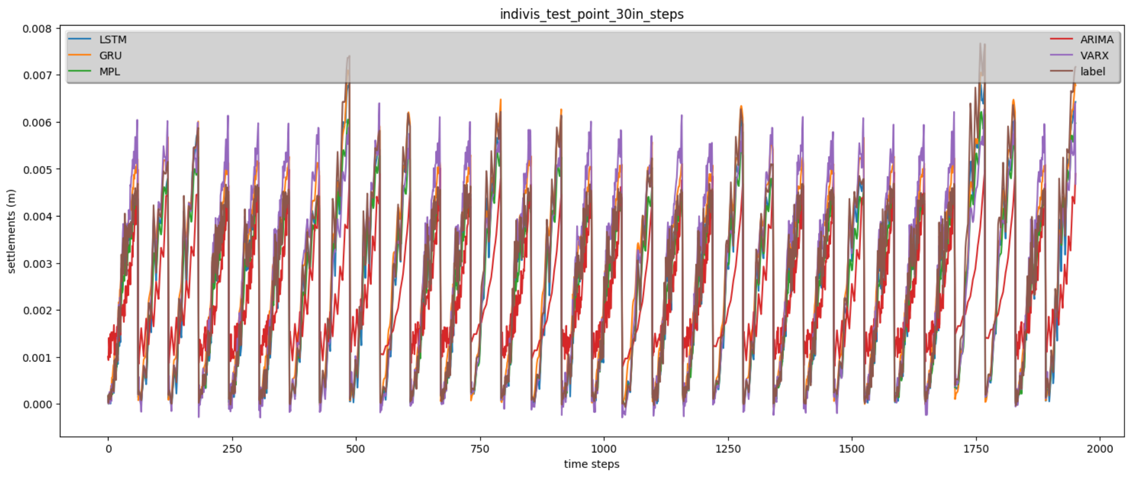


Figure A.6: Meta-learner result in test phase and the label of point 38

A.0.2 Meta-Learner

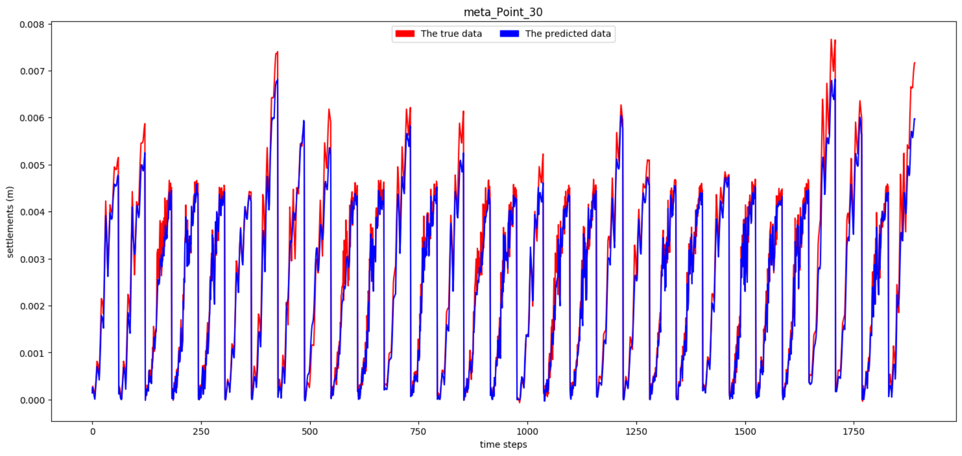


Figure A.7: Meta-learner result in test phase and the label of point 30

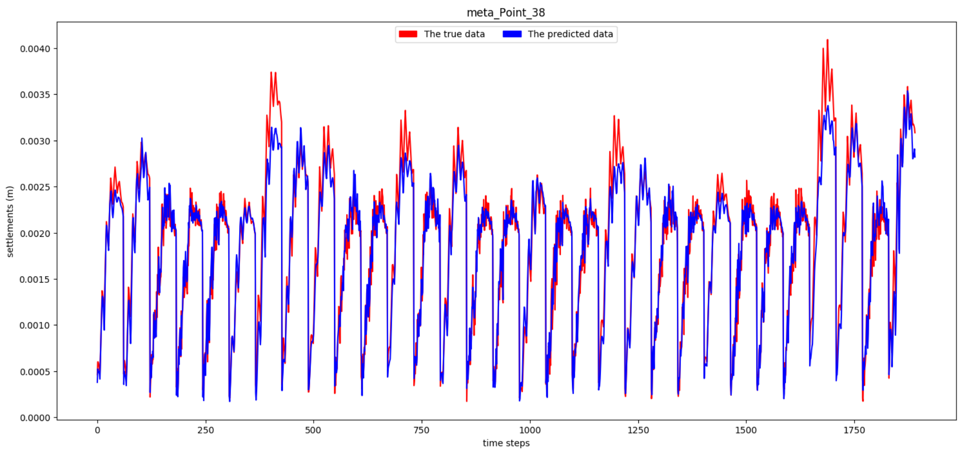


Figure A.8: Meta-learner result in test phase and the label of point 38

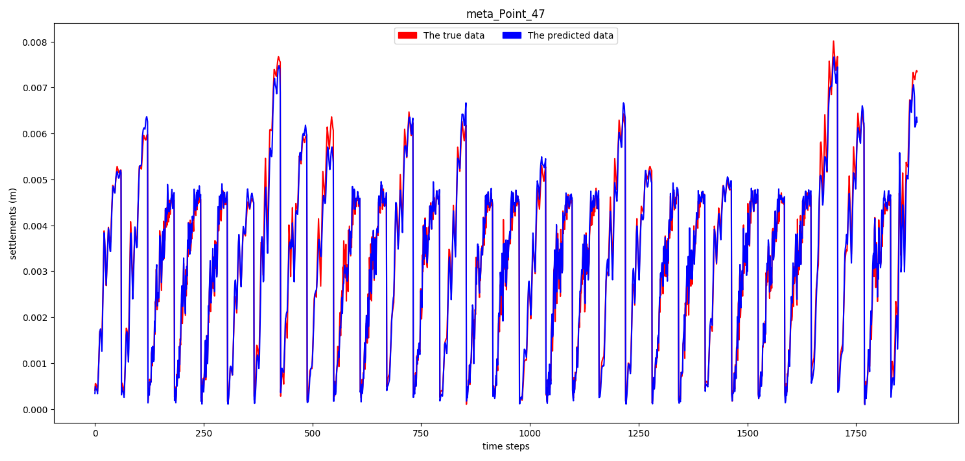


Figure A.9: Meta-learner result in test phase and the label of point 47

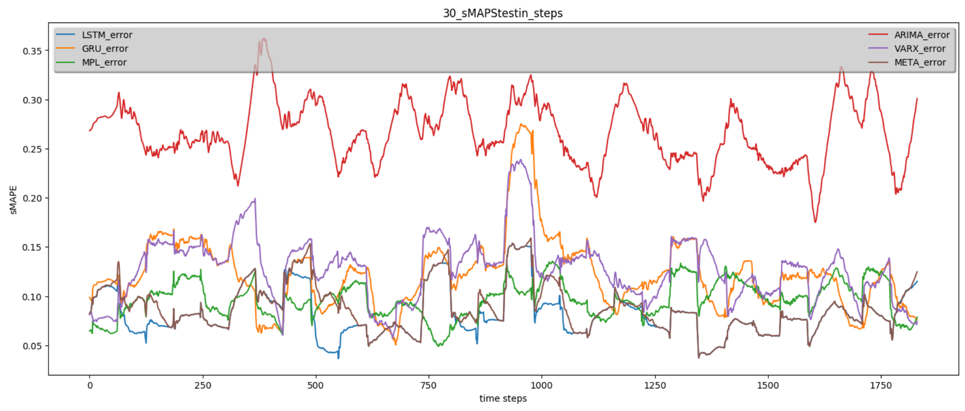


Figure A.10: individuals and meta-learner error in test phase of point 30

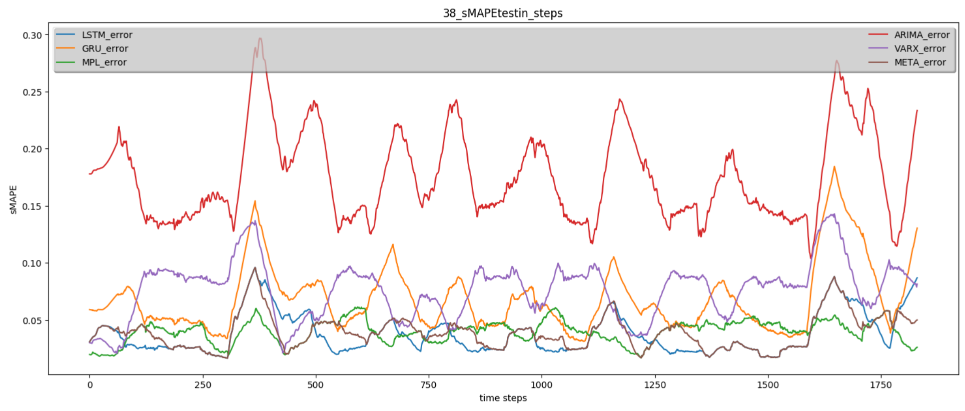


Figure A.11: individuals and meta-learner error in test phase of point 38

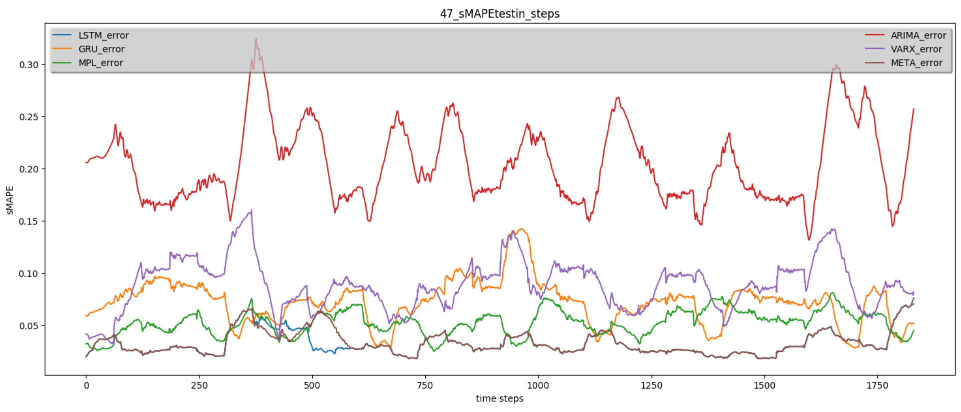


Figure A.12: individuals and meta-learner error in test phase of point 47

A.0.3 Meta-learner for ensemble models selection

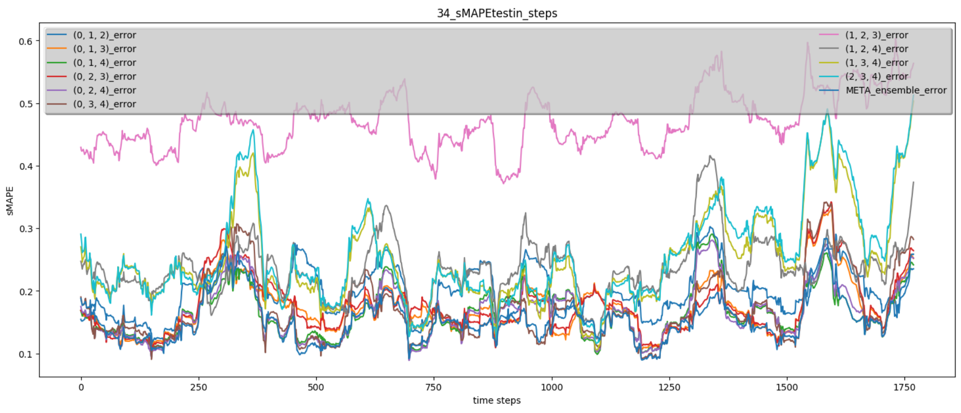


Figure A.13: meta learner selected ensemble error in test phase of point 34

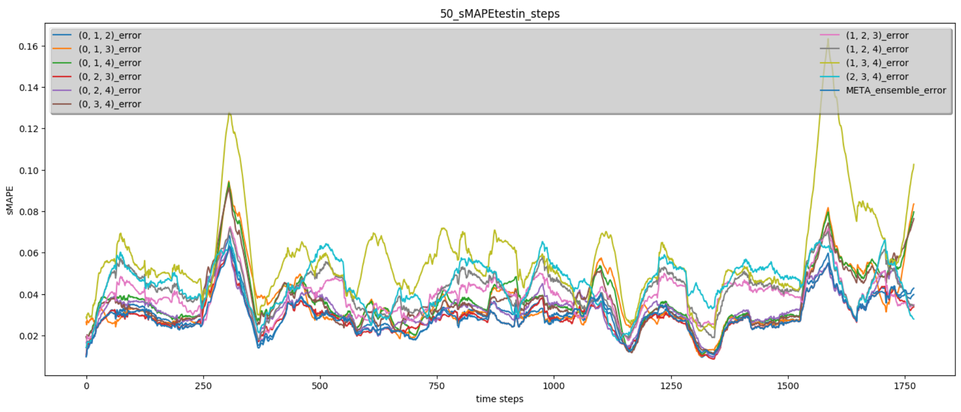


Figure A.14: meta learner selected ensemble error in test phase of point 50

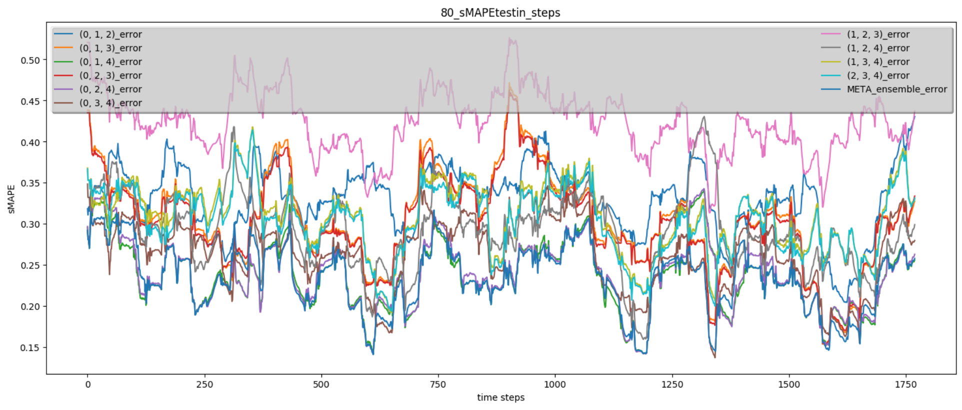


Figure A.15: meta learner selected ensemble error in test phase of point 80

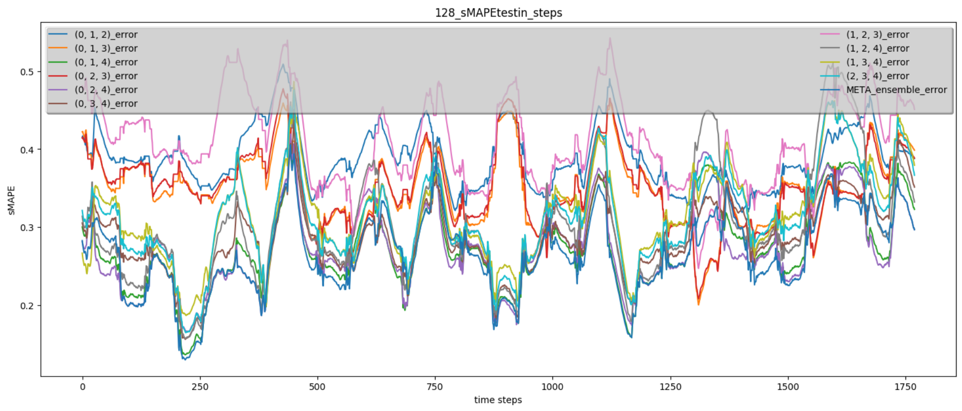


Figure A.16: meta learner selected ensemble error in test phase of point 128

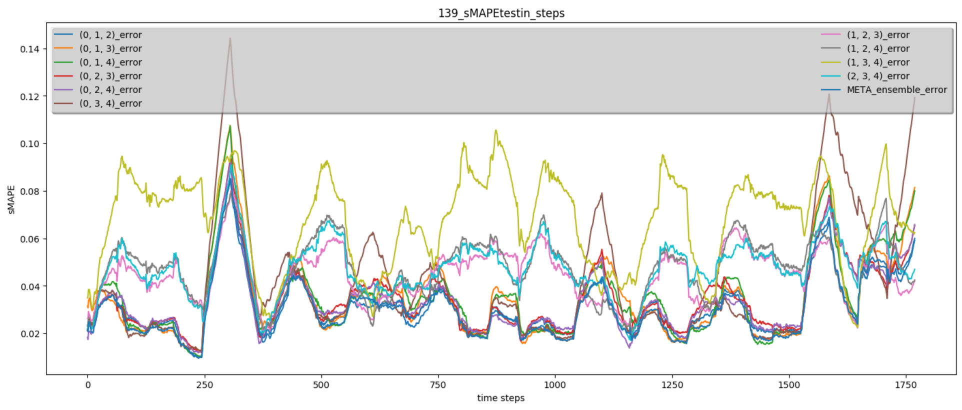


Figure A.17: meta learner selected ensemble error in test phase of point 139

	MAPE	**sMAPE**	**RSME**	**NRSME**
LSTM	31.10% ± 41.24%	16.24% ± 12.16%	0.04% ± 0.03%	4.80% ± 1.89%
GRU	115.76% ± 184.02%	30.02% ± 19.40%	0.09% ± 0.08%	11.81% ± 4.72%
MLP	70.66% ± 110.51%	23.71% ± 16.16%	0.08% ± 0.08%	9.42% ± 3.42%
ARIMA	676.01% ± 1189.73%	39.81% ± 16.94%	0.15% ± 0.14%	18.08% ± 4.64%
VARX	128.94% ± 224.40%	15.81% ± 8.04%	0.05% ± 0.03%	7.07% ± 2.40%
ensemble(0, 1, 2)	39.02% ± 48.89%	16.47% ± 13.07%	0.04% ± 0.03%	4.96% ± 1.81%
ensemble(0, 1, 3)	36.93% ± 49.77%	15.55% ± 11.73%	0.04% ± 0.03%	5.35% ± 2.22%
ensemble(0, 1, 4)	39.70% ± 47.58%	12.26% ± 8.25%	0.03% ± 0.02%	4.45% ± 1.52%
ensemble(0, 2, 3)	35.61% ± 43.98%	15.04% ± 11.86%	0.04% ± 0.03%	4.90% ± 1.69%
ensemble(0, 2, 4)	39.56% ± 46.59%	12.18% ± 8.24%	0.03% ± 0.02%	4.22% ± 1.15%
ensemble(0, 3, 4)	37.76% ± 43.74%	12.63% ± 9.09%	0.03% ± 0.02%	4.52% ± 1.55%
ensemble(1, 2, 3)	67.84% ± 96.76%	22.07% ± 15.26%	0.08% ± 0.08%	9.25% ± 3.25%
ensemble(1, 2, 4)	69.24% ± 97.72%	14.78% ± 9.26%	0.04% ± 0.03%	5.88% ± 1.52%
ensemble(1, 3, 4)	85.79% ± 133.13%	15.52% ± 9.46%	0.05% ± 0.03%	6.54% ± 2.01%
ensemble(2, 3, 4)	72.38% ± 106.78%	15.04% ± 9.74%	0.04% ± 0.03%	5.99% ± 1.46%
Selected Ensemble	42.96% ± 52.36%	15.17% ± 11.75%	0.04% ± 0.03%	5.07% ± 2.13%
Meta leaner Individual	34.64% ± 45.79%	14.79% ± 9.54%	0.03% ± 0.02%	4.71% ± 1.71%
Meta learner Ensemble	25.67% ± 23.57%	11.37% ± 7.99%	0.03% ± 0.02%	4.09% ± 1.14%

Table A.1: overage errors over all 154 points in the test phase, where the number 0,1,2,3,4 indicate the model LSTM, GRU, MLP, ARIMA, VARX respectively

List of Figures

Bibliography

[1] Beveridge, Stephen and Charles R Nelson: *A new approach to decomposition of economic time series into permanent and transitory components with particular attention to measurement of the 'business cycle'.* Journal of Monetary economics, 7(2):151–174, 1981.

[2] Bierens, Herman J: *VAR models with exogenous variables.* Available on http://econ. la. psu. edu/~ hbierens/EasyRegTours/VAR_Tourfiles/VARX. PDF, 2004.

[3] Chitra, A and S Uma: *An ensemble model of multiple classifiers for time series prediction.* International Journal of Computer Theory and Engineering, 2(3):454, 2010.

[4] Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio: *Learning phrase representations using RNN encoder-decoder for statistical machine translation.* arXiv preprint arXiv:1406.1078, 2014.

[5] Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio: *Empirical evaluation of gated recurrent neural networks on sequence modeling.* arXiv preprint arXiv:1412.3555, 2014.

[6] Contreras, Javier, Rosario Espinola, Francisco J Nogales and Antonio J Conejo: *ARIMA models to predict next-day electricity prices.* IEEE transactions on power systems, 18(3):1014–1020, 2003.

[7] Dekimpe, Marnik G and Dominique M Hanssens: *Time-series models in marketing:: Past, present and future.* International journal of research in marketing, 17(2-3):183–193, 2000.

[8] Ganeshan, Kala and D Joseph Anbarasu2 P Senthil Kumar: *A Study on ARIMA and ARMA Models of Selected Shares in NSE and BSE during the Period of Financial Crisis.* 2014.

[9] Graves, Alex, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke and Jürgen Schmidhuber: *A novel connectionist system for unconstrained handwriting recognition.* IEEE transactions on pattern analysis and machine intelligence, 31(5):855–868, 2009.

[10] Hansen, Lars Kai and Peter Salamon: *Neural network ensembles.* IEEE transactions on pattern analysis and machine intelligence, 12(10):993–1001, 1990.

[11] Hochreiter, Sepp and Jürgen Schmidhuber: *Long short-term memory.* Neural computation, 9(8):1735–1780, 1997.

[12] Kingma, Diederik P and Jimmy Ba: *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980, 2014.

[13] Ko, Albert HR, Robert Sabourin and Alceu Souza Britto Jr: *From dynamic classifier selection to dynamic ensemble selection.* Pattern Recognition, 41(5):1718–1731, 2008.

[14] Krikunov, Alexey V and Sergey V Kovalchuk: *Dynamic Selection of Ensemble Members in Multi-Model Hydrometeorological Ensemble Forecasting.* Procedia Computer Science, 66:220–227, 2015.

[15] Kück, Mirko, Sven F Crone and Michael Freitag: *Meta-learning with neural networks and landmarking for forecasting model selection an empirical evaluation of different feature sets applied to industry data.* In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 1499–1506. IEEE, 2016.

[16] Lani, James: *Time series analysis*, 1994.

[17] Lütkepohl, Helmut: *New introduction to multiple time series analysis.* Springer Science & Business Media, 2005.

[18] Moreira-Matias, Luis, Joao Gama, Michel Ferreira, Joao Mendes-Moreira and Luis Damas: *Predicting taxi–passenger demand using streaming data.* IEEE Transactions on Intelligent Transportation Systems, 14(3):1393–1402, 2013.

[19] Ocampo, Sergio and Norberto Rodríguez: *An introductory review of a structural VAR-X estimation and applications.* Revista Colombiana de Estadística, 35(3):479–508, 2012.

[20] Prudêncio, Ricardo BC and Teresa B Ludermir: *Meta-learning approaches to selecting time series models.* Neurocomputing, 61:121–137, 2004.

[21] Sak, Haşim, Andrew Senior and Françoise Beaufays: *Long short-term memory recurrent neural network architectures for large scale acoustic modeling.* In *Fifteenth annual conference of the international speech communication association*, 2014.

[22] Wang, Haixun, Wei Fan, Philip S Yu and Jiawei Han: *Mining concept-drifting data streams using ensemble classifiers.* In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235. AcM, 2003.

[23] Wang, Xiaozhe, Kate Smith-Miles and Rob Hyndman: *Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series.* Neurocomputing, 72(10-12):2581–2594, 2009.

[24] Xingjian, SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong and Wang-chun Woo: *Convolutional LSTM network: A machine learning approach for precipitation nowcasting.* In *Advances in neural information processing systems*, pages 802–810, 2015.

[25] Zaremba, Wojciech, Ilya Sutskever and Oriol Vinyals: *Recurrent neural network regularization.* arXiv preprint arXiv:1409.2329, 2014.

[26] Zhang, G Peter: *Time series forecasting using a hybrid ARIMA and neural network model.* Neurocomputing, 50:159–175, 2003.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den December 2, 2018

Donghui He