FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Fusion of heterogeneous models in agent networks
Student:	Bc. Martin Šmíd
Supervisor:	Ing. Kamil Dedecius, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

Assume a network of agents observing and modelling some stochastic process with the goal to predict its future or intermediate values. The agents are independent in the sense that they are allowed to locally employ different models, for instance, physical, time-series, state-space, black or white box models. The questions are the following: Can the agents improve their local predictions by considering predictions of neighbouring nodes? How to incorporate this information into nodes' own local statistical knowledge about the predicted variables?

The main points of the thesis are:
1. Study the topic of collaborative modelling in networks of interconnected agents.
2. Propose a method for improving local predictions by exploiting information provided by neighbouring nodes. In particular, what information should be exchanged and how to combine it locally.
3. Demonstrate the proposed method properties on convenient simulation and/or real-world examples.
p { margin-bottom: 0.1in; line-height: 115%; background: transparent none repeat scroll 0% 0%; }

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague November 5, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Fusion of heterogeneous models in agent networks

Bc. Martin Šmíd

Department of Applied Mathematics
Supervisor: Ing. Kamil Dedecius, Ph.D.

June 1, 2020

Acknowledgements

I would like to thank everyone who supported me during my studies, especially my family, friends, and special friends. It would be very difficult for me without any single one of them. I would also like to thank my supervisor for his guidance and patient help.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on June 1, 2020 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Šmíd, Martin. *Fusion of heterogeneous models in agent networks.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato práce se zabývá možnostmi zlepšení lokálních predikcí v agentních sítích pomocí sdílení informací se sousedními uzly. Agenti používají různé modely, v důsledku čehož je omezen typ informací, které si mohou vyměnit se sousedy. Je navrženo několik variant ke zlepšení predikce, některé z nich umožňují zahrnutí nejistoty predikce. Varianty jsou následně implementovány a porovnány na simulovaných i reálných datech.

Klíčová slova kolaborativní modelování, heterogenní modely, difúzní sítě

Abstract

This thesis looks into the ways of enhancing the local predictions in agent networks using the information shared by their neighbouring nodes. Agents locally employ different models, thus the type of information to share is limited. Multiple variants of improving the predictions are proposed, some of them taking the prediction uncertainty into consideration. These are then implemented and compared on both simulated and real-world data.

Keywords collaborative modelling, heterogeneous models, diffusion networks

Contents

List of Figures

List of Tables

Chapter 1

State of Art

Mathematical modelling is everywhere. It is absolutely necessary in multiple fields of human activity – physics, meteorology, biology, chemistry, economics, to name but a few.

The aim of modelling is typically to estimate hidden variables with or without a clear and straightforward interpretation, to predict the future values, or to interpolate the past ones.

Probabilistic and statistical modelling is one of the branches of modelling. It infers the system behaviour based on assumptions of the observed variables.

We face the issue of choosing a suitable model reflecting the reality. However, multiple such models exist and it is often questionable to select only one of them based on the observed data. That is why several approaches emerged throughout the history. Bagging methods use data sampling to provide models with different subset of data [1]. Boosting methods sequentially improve the model and then use their combination [2]. Bayesian model averaging takes the prediction uncertainties of individual models into account [3], [4].

In the last decades, small devices with high computational ability are more and more prevalent. Many applications arise with the same variable of interest modelled by multiple independent devices. Their mutual connection can be used to exchange information in attempt to improve the modelling process instead of being only the information collectors.

Distributed estimation of parameters of the same models is rather extensive topic [5], [6]. It can use one of the several communication protocols – namely incremental, consensus with iterations, and diffusion without iterations, or its variations [7], [8].

This thesis focuses on the rather omitted issue of collaborative predictions in case that the individual agents employ different models and therefore the parameter estimation is not to consider. The fusion of the predictions is partially inspired by the adaptation and combination steps in diffusion networks having low communication overhead, where it serves to estimate the model parameters [7]. Further inspiration is found in Bayesian model averaging and

its variants [3], [9].

Thesis structure

This project aims to propose, implement, and test some methods of collaboration among agents which would improve the agents' local predictions of some observed process.

The purpose of Chapter 2 is to look into general ideas regarding the combination of multiple models. Next, collaboration between models is discussed and some of the approaches to the issue are briefly described.

The objective of Chapter 3 is to propose the methods improving the local predictions of the agents. General overview of the information exchange between agents and subsequent incorporation into the agent's prediction used later on is described in this part.

In Chapter 4, implementation is presented. Software necessary and its usage is listed and commented on. Specific details and possibilities of the implemented variants are described.

The objective of Chapter 5 is to test the implemented solution, compare the proposed variants, discuss possible extensions to the variants proposed, and, last but not least, mention other approaches.

Terms and expressions used are listed in the Appendix A.

Chapter 2

Preliminary theory

This chapter presents several theoretical concepts related to combinations of models. Possible evaluation of models is briefly mentioned. Finally, distributed modelling is described.

2.1 Model

Model is defined as a description of certain system. Purpose of (mathematical) modelling is to explain the behaviour of the system, infer the characteristics of the system, and possibly predict its future behaviour.

Models can be categorized in multiple ways depending on the properties studied. One of the important attributes is time-relevancy – static models describe the systems constant with time, whereas dynamic models reflect the changes in the state of the system. In particular, this thesis focuses on parametric statistical models. Statistical models comprise of statistical assumptions of how the sample data was generated. They are usually described as a pair $(S, \mathcal{P})$, where S is a set including possible observations and $\mathcal{P}$ is a set of probability distributions on the possible observations. A parametric statistical model can be thought of as a model with finite number of parameters that describe the model.

2.2 Individual models

2.2.1 Autoregressive (AR) model

Let Y_t, $t = 1, 2, \ldots$ be a time series. Autoregressive model of order p, abbreviated as AR(p), is described as

$$Y_t = c + \phi_1 Y_{t-1} + \ldots + \phi_p Y_{t-p} + \varepsilon_t = c + \sum_{\tau=1}^{p} \phi_\tau Y_{t-\tau} + \varepsilon_t,$$

where Y_t is real and for simplicity scalar random variable, c is offset, $\phi_1, \ldots, \phi_p$ are regression coefficients, and $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ is white noise.

2.2.2 Moving average (MA) model

Let Y_t, $t = 1, 2, \ldots$ be a time series. Moving average model of order q, abbreviated as MA(q), is described as

$$Y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q},$$

where Y_t is real and for simplicity scalar random variable, μ is the expected value of Y_t, $\theta_{1..p}$ are MA coefficients, and $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ is white noise.

2.2.3 ARIMA model

Autoregressive integrated moving average model ARIMA(p,d,q) is a mixed combination of AR(p) and MA(q) models, applied to d-times differentiated series obtained by subtracting the consequent observations

$$Y'_t = Y_t - Y_{t-1},$$

where Y_t and Y_{t-1} are real and for simplicity scalar random variable. ARIMA models can be very powerful, with d=1 or d=2 usually applied. The differentiation is used to suppress the trend in the data.

2.2.4 SARIMA model

SARIMA(p,d,q)(P,D,Q)s is an ARIMA(p,d,q) model with added seasonal component ARIMA(P,D,Q), which is repeated s times in a year. It is used to describe the data if periodicity is assumed.

2.2.5 Prophet model

Prophet is a time series model developed by Facebook. It employs three main components in the model – trend, seasonality, and holidays. Therefore the Prophet model is able to possibly reflect the impact of the calendar day in the time series, which can be useful in real-life modelling. [10]

2.2.6 Exponential smoothing

Exponential smoothing is a technique similar to MA. However, it assigns the past observations exponentially decreasing weights with time. It is described as

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1} = s_{t-1} + \alpha(x_t - s_{t-1}),$$

where α is the smoothing factor such that $0 < \alpha < 1$ and s_t is the smoothed value at time t. These smoothed values can be considered as a prediction of the next value, that is $Y_t = s_t$.

2.3 Model selection criteria

When several statistical models are considered, a selection criterion can help to choose the more appropriate model in comparison with the others. Akaike information criterion (AIC) and Bayesian information criterion (BIC) are commonly used.

2.3.1 AIC

AIC can be defined in the following way:

$$AIC = 2k - 2\ln \mathcal{L},$$

where $\mathcal{L}$ is the maximized likelihood function of the model and k is the number of estimated model parameters. Lower values of AIC imply a better model. Therefore, if two models fit the model equally well, according to AIC, the better model is the one with the lower number of independently adjusted model parameters. [11]

2.3.2 BIC

BIC can be defined in the following way:

$$BIC = \ln(n)k - 2\ln \mathcal{L},$$

where $\mathcal{L}$ is the maximized likelihood function of the model, k is the number of estimated model parameters, and n is the number of observations. Incorporation of the number of observed samples can prevent preference of the overfitted models. [12], [13]

2.4 Combining models

This part presents several methods of model combination in non-distributed modelling.

2.4.1 Standard approach

Bagging

The idea of bagging methods is to create multiple models $M_1, \ldots, M_K$, each fitting a distinct subset of the available data sampled uniformly with replacement (meaning that the observations used for individual models might be repeated). The models are then combined – the prediction is made as the average (or possibly another combination) of the predicted values of $M_1, \ldots, M_K$. [1]

Boosting

The idea of boosting methods is to sequentially update the model and thus creating multiple models $M_1, \ldots, M_K$. The model is updated in the manner that it is gradually provided with the data previously handled the worst. The models are then combined – the prediction is made as the average (or possibly another combination) of the predicted values of $M_1, \ldots, M_K$. [2]

2.4.2 Bayesian approach

Bayesian approach is to incorporate the uncertainties into the model combination.

Bayesian model averaging (BMA)

Bayesian model averaging (BMA) is a static method to combine the models. In case multiple models seem to fit the data comparably well, but provide different predictions, model selection resulting in only one chosen model may not be easy and appropriate.

Suppose that there is a set of possible candidate models $M_1, \ldots, M_K$, each modelling data $D = \{y_0, \ldots, y_{t-1}\}$ with a probability density function $p(y_t \mid M_k, D)$. Then, we aim at deciding probabilities $p(M_k \mid D)$ quantifying the likelihood that the particular data D were generated according to each M_k.

Posterior distribution of y_t (the matter of interest) is

$$p(y_t \mid D) = \sum_{k=1}^{K} p(y_t \mid M_k, D) p(M_k \mid D),$$

where M_k are the models taken into consideration and are weighted by their posterior model probability. Posterior probability for model M_k is

$$p(M_k \mid D) = \frac{p(D \mid M_k) p(M_k)}{\sum_{l=1}^{K} p(D \mid M_l) p(M_l)},$$

where integrated likelihood of model $p(D \mid M_k)$ is

$$p(D \mid M_k) = \int p(D \mid \theta_k, M_k) p(\theta_k \mid M_k) d\theta_k,$$

where θ_k is the vector of parameters of model M_k. Finally, the posterior mean and variance of the modelled variable y_t are

$$E[y_t \mid D] = \sum_{k=0}^{K} \hat{y}_{tk} p(M_k \mid D)$$

$$Var[y_t \mid D] = \sum_{k=0}^{K} (Var[y_t \mid D, M_k] + \hat{y}_{tk}^2) p(M_k \mid D) - E[y_t \mid D]^2,$$

where $\hat{y}_{tk}^2 = E[y_t \mid D, M_k]$. [3]

Dynamic model averaging (DMA)

Dynamic model averaging (DMA) is a dynamic method to combine the individual models $M_1, \ldots, M_K$. It is thus used in online predictions, where data arrives sequentially.

DMA is a BMA extension, enabling the variation of both the regression model and regression parameters with time. If one of the considered models turn up to describe the data far better than the other models, DMA converges to this model. That results in not getting worse predictions (which would be caused only by the uncertainty related to this best model). [14], [15], [16]

2.5 Distributed modelling

2.5.1 Network properties

The network is represented by a graph composed of N nodes representing the agents and by a set of edges creating the connections among the agents. The neighbourhood of agent k consists of all the agents that are connected to k by an edge, including k. Closed neighbourhood, denoted $\mathcal{N}_k$, includes agent k, open neighbourhood excludes k. Any two agents connected by an edge are considered *neighbours* and can share information with each other. Cardinality of the closed neighbourhood $\mathcal{N}_k$ is equal to the number of distinct nodes in $\mathcal{N}_k$ and is denoted $|\mathcal{N}_k|$.

2.5.2 Communication

Communication in the network can be centralized or decentralized. If centralized communication is used, a fusion center exists wherein the data is processed, and then possibly sent back to the agents. It can be a problem in some cases, typically when the network is large and the emphasis is put on the real-time processing. If sharing the data is a privacy issue or security concerns pose a threat, centralized approach is also disqualified. Moreover, centralized approach obviously represents a single point of failure – for those reasons, decentralized communication is assumed in many practical applications, that is, agents communicate with several neighbours on a peer-to-peer level. [7], [8]

2.5.3 Strategies of learning in multi-agent networks

Three main strategies exist in order to distribute the modelling (or, generally, minimize the error) in the network:

- **incremental** – the information is passed from node to node in a Hamiltonian cycle. This solution is quite robust with respect to information processing. However, each node is a single point of failure and potential recovery from a failure is an NP-hard problem. [7] Figure 2.1 depicts

the incremental strategy on the graph consisting of 8 nodes. Node 1 is highlighted in red, along with its connections to neighbouring nodes.

- **consensus** – the network nodes share the information with their adjacent neighbours, forming the closed neighbourhood. The communication and optimization runs in several iterations. [7]
- **diffusion** – the principle is similar to the consensus strategy, but the information is exchanged only once, without any intermediate iterations. The exchange comprises either observations (so-called adaptation) and/or estimates (combination step). Based on the order of the steps, the diffusion strategies are either ATC (adapt-then-combine) or CTA (combine-then-adapt). [7] Figure 2.2 depicts the consensus/diffusion strategy on the example graph of 8 nodes. Node 1 is highlighted in red, along with its connections to neighbouring nodes.

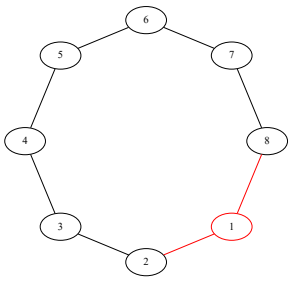

Figure 2.1: Illustration of incremental strategy in the network.

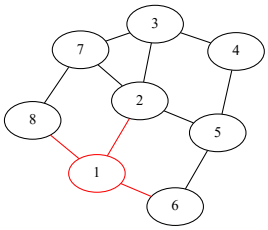

Figure 2.2: Illustration of consensus strategy in the network.

Chapter 3

Proposed solution

This chapter aims to propose a general approach to the issue. Due to their heterogeneity and diverse inner functioning, the models are to a large extent considered black-boxes. That results in the need of using a general, universal approach to gather and process the data they can provide to improve the local predictions.

3.1 Description of the issue

Firstly, it is important to lay down the problem. Suppose a network of heterogeneous agents observing and modelling a stochastic process. The main questions posed are:

- Can the agents improve their local predictions by considering predictions of neighbouring nodes?
- How to incorporate predictions of neighbouring nodes into nodes' own local knowledge about the predicted variables?

3.2 Methods proposed

This work proposes a general method to combine the predictions (one time step ahead) in distributed manner. Diffusion protocol served as the inspiration because of its low communication overhead. The predictions are fused using different simple approaches. The following lines describe some variants that could be used to improve local predictions by using information from neighbouring nodes. That is inspired by carrying out the *combination* part of the diffusion strategy and is applied only for current prediction. Variants 0, 1, 2, 3, and 4 are implemented and compared in Chapters 4 and 5.

3.2.1 General

The following parameters are generally varied in any of the methods proposed (variant 0, variant 1, variant 2, variant 3, and variant 4):

- node degree of the network (number of neighbours)
- network size
- sharing or not of the observed value of the process – that is inspired by carrying out the *adaptation* part of the diffusion strategy

If sharing of the observed value is performed, each node i uses the mean value of the values $\hat{y}_t(k)$ obtained from its closed neighbourhood $\mathcal{N}_i$ instead of the observed value

$$y_t(i) = \frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} \hat{y}_t(k),$$

where $\hat{y}_t(k)$ are the original observed values from $\mathcal{N}_i$ and $y_t(i)$ is the new, updated observation.

Local mean squared error (MSE) of inner model of agent i after t predictions evaluated by agent m is defined as

$$\mathrm{MSE}_{mi} = \frac{1}{t} \sum_{\tau=1}^{t} (y_\tau(m) - \hat{y}_\tau(i))^2,$$

where $y_\tau(m)$ is the value observed by agent m at time τ (possibly updated by the observations of neighbouring agents if sharing of observations is applied) and $\hat{y}_\tau(i)$ is the inner prediction of agent i at time τ.

Similarly, local mean absolute error (MAE) of inner model of agent i after t predictions evaluated by agent m is

$$\mathrm{MAE}_{mi} = \frac{1}{t} \sum_{\tau=1}^{t} \left| (y_\tau(m) - \hat{y}_\tau(i)) \right|,$$

where $y_\tau(m)$ is the value observed by agent m at time τ (possibly updated by the observations of neighbouring agents if sharing of observations is applied) and $\hat{y}_\tau(i)$ is the inner prediction of agent i at time τ.

3.2.2 Weights

Two of the variants proposed use local weights of nodes in $\mathcal{N}_i$ to combine the predictions. Local weight evaluated in node i of node k before normalization is

$$w_{ik_{tmp}} = 1 - \frac{err_k}{\sum_{m \in \mathcal{N}_i} err_m},$$

where err_k, $err_m, m \in \mathcal{N}_i$ are the errors of corresponding agents k or m equal to MSE_{ik} or MSE_{im} respectively. MAE (or possibly other error functions) can also be applied.

After normalization (in order to scale the predictions properly), we get

$$w_{ik} = \frac{w_{ik_{tmp}}}{\sum_{m \in \mathcal{N}_i} w_{im_{tmp}}},$$

where the weight before normalization is divided by the sum of all the weights before normalization in the agent's closed neighbourhood $\mathcal{N}_i$.

3.2.3 Variant 0 – *ignore*

This variant, simple no collaboration scenario, is used as a comparison and is considered the worst variant – each agent accounts for predictions of its inner model only and ignores the predictions of the neighbouring nodes. In case of competing variants being unable to outperform the accuracy of this variant 0, that would signify that those variants only produce useless communication and computation overhead without any contribution.

This variant is also referred to as variant *ignore*.

3.2.4 Variant 1 – *best_neighbour*

This variant is a simple model selection method. Each agent shares its prediction with neighbours and then selects the best prediction, based on the past accuracy (evaluated by MSE or MAE). This approach should be better than the variant 0, but it is really simplistic. On the other hand, its implementation is rather trivial.

The prediction $y_{t+1}(i)$ of agent i is thus

$$y_{t+1}(i) = \hat{y}_{t+1}(k),$$

where $\hat{y}_{t+1}(k)$ are the predictions of inner models for $k \in \mathcal{N}_i$ and $w_k >= w_m$ for $m \in \mathcal{N}_i$. If two or more such nodes exist, the arithmetic mean of their inner predictions $\hat{y}_{t+1}(k)$ should be used instead.

This variant is also referred to as variant *best_neighbour*.

3.2.5 Variant 2 – *average*

This variant also employs a simple approach – the prediction is made as arithmetic mean of agent's neighbours (agent itself included). This approach could be prone to outliers in some cases.

The prediction $y_{t+1}(i)$ of agent i is

$$y_{t+1}(i) = \frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} \hat{y}_{t+1}(k),$$

where $\hat{y}_{t+1}(k)$ are the predictions of inner models for $k \in \mathcal{N}_i$.

In multiple real-life scenarios, it was empirically shown that simple arithmetic mean of point predictions of the models can outperform more complicated methods using weighting the predictions based on the MSE. [17] That is why this variant is also implemented and compared.

This variant is also referred to as variant *average*.

3.2.6 Variant 3 – *weighted_average*

Another variant proposed calculates the "weights" of the nodes in agent's closed neighbourhood. Each agent then determines its prediction as a dot product of the neighbours' weights and the corresponding predictions (again, closed neighbourhood is supposed, so the prediction of agent's inner model is taken into account).

The combined prediction $y_{t+1}(i)$ of agent i is

$$y_{t+1}(i) = \sum_{k \in \mathcal{N}_i} \hat{y}_{t+1}(k) w_{ik},$$

where $\hat{y}_{t+1}(k)$ are the inner predictions of inner models for $k \in \mathcal{N}_i$.

This variant is also referred to as variant *weighted_average*.

3.2.7 Variant 4 – *weighted_median*

The last variant proposed is very similar to the variant 3 mentioned above, *weighted_average*. The only difference is in the final prediction – it is not the dot product of the neighbours' weights and the corresponding predictions, but weighted median of those values. Therefore, one of the predictions in agent's closed neighbourhood is chosen in the manner that the difference between the dot products of the two parts divided by the median value is the least.

Supposing ordered inner model predictions $\hat{y}_{t+1}$ of agents in $\mathcal{N}_i$, the combined prediction $y_{t+1}(i)$ of agent i is

$$y_{t+1}(i) = \hat{y}_{t+1}(k)$$

such that

$$\sum_{p=1}^{k-1} w_{ip} \leq \frac{1}{2} \bigwedge \sum_{p=k+1}^{|\mathcal{N}_i|} w_{ip} \leq \frac{1}{2}$$

That means the "middle" value (median) is selected with regard to models weights. If two values $\hat{y}_{t+1}(k)$ satisfy this condition, their arithmetic mean is considered as the combined prediction $y_{t+1}(i)$.

This variant is also referred to as variant *weighted_median*.

3.2.8 Certainties in variants 1, 3, 4

Not dissimilar to Bayesian model averaging, several models can supply the others with information describing the extent of belief in the prediction ("certainty"), which is then categorized as one of the values in a vector. This vector ("mask"), such as [0.95, 1.0, 1.05], will be used with mean equal to one and rather small variance – so that the weights are tuned conservatively. This *a priori* mask is fixed throughout the experiment and transfers the responsibility of determining the (un)certainty of its own prediction to each agent.

The advantage of this approach is the abstraction, independent on the model – either the model supports it, or a default value of the mask [1.0] is used. Another advantage is that the mask can be specifically tuned for each agent separately, especially if model accuracy is known (or guessed) in advance. The possible disadvantage is that the realisation and its results are highly dependent on the type of the model and the quality of determining the certainty.

In practice, this will be tested using the negative value of BIC (or AIC in several cases) of the model as the certainty

$$cert_{non-Prophet} = -\text{BIC}$$

In case of Prophet model, the certainty is determined based on the spread of the prediction interval

$$cert_{Prophet} = 1 - \left| \frac{y_{high} - y_{low}}{y_{pred}} \right|,$$

where y_{high} is the higher value of the boundary of the prediction interval, y_{low} is the lower value of the boundary of the prediction interval, and y_{pred} is the predicted value.

After the certainty is calculated, it must be mapped on the values of the mask. That is accomplished by the past certainty values – according to its probability distribution, it is binned into the quantile corresponding to the mask. For example, if the given past model certainties were [4, 5, 6, 7, 8] and the mask was [0.95, 1.00, 1.05]:

- the value 4.5 would be masked to 0.95
- the value 5.3 would be masked to 1.00
- the value 6.9 would be masked to 1.05

Binned certainty values are distributed to the neighbouring nodes, where they serve as weight modifiers. They are simply multiplied

$$w_{cert_i} = w_i c_i,$$

where w_i is local weight of node i in closed neighbourhood and c_i is binned prediction certainty corresponding to the node.

In order to determine the agent's prediction, the normalized weight is required

$$w_{n_i} = \frac{w_{cert_i}}{\sum_{k \in \mathcal{N}_i} w_{cert_k}},$$

where normalized weight w_{n_i} is calculated as a fraction – corresponding weight modified by certainty w_{cert_i} is divided by the sum of the weights modified by certainty of all agents $k \in \mathcal{N}_i$. Then, these modified weights are used instead of the "standard" weights in the variants if applicable.

3.3 Advantages

Proposed variants have multiple advantages:

- abstract view of the models
- individual variants are straightforward in their interpretation
- most of the variants can be combined with using certainties, the agents' predictions thus take into account discretized certainties of the nodes in the closed neighbourhood (in case the agents support the certainty calculation)
- low computational cost – in comparison with other approaches, the computational burden of the variants is rather low (although it may be higher if the certainty calculation is demanding)

3.4 Other possibilities

The disadvantage of the proposed variants is not considering all the information some of the models may provide. If Bayesian approach is applied, the whole predictive distributions could be combined instead of discretized values.

Other possible conceptions not implemented are mentioned and roughly described in Sections 5.7 and 5.8.

Chapter 4

Implementation

Objective of this chapter is to present and describe the implementation of the variants laid out in previous chapter. Fairly detailed description of the procedure of running the developed code is given. Selected parts of inner workings of the implementation are also described, along with their possibilities and limitations.

4.1 Software

This section lists software necessary to run the implementation. Predominantly, this applies for:

- Python and its packages
- Anaconda, especially on Windows OS
- Jupyter

4.1.1 Python

In some cases, because of mutual package dependencies, problems could arise while using different versions of packages, though the newest versions should work. For convenience, packages and libraries used to process and model the data are listed, in alphabetical order:

- fbprophet – version 0.6
- numpy – version 1.18.1
- pandas – version 1.0.1
- robustats – version 0.1.5
- scikit-learn – version 0.22.1
- scipy – version 1.4.1

- statsmodels – version 0.11.0

More info regarding the usage and capabilities of these packages can be found at: [10], [18], [19], [20], [21], [22], [23].

For visualization, packages used include graphviz (in version 2.38) and matplotlib (in version 3.1.3). More reference can be found at: [24], [25].

Python itself is used in version 3.7.6.

4.1.2 Jupyter

Versions of important Jupyter [26] components used are listed below:

- jupyter core – version 4.6.1
- jupyter-notebook – version 6.0.3
- ipython – version 7.12.0
- ipykernel – version 5.1.4
- jupyter client – version 5.3.4

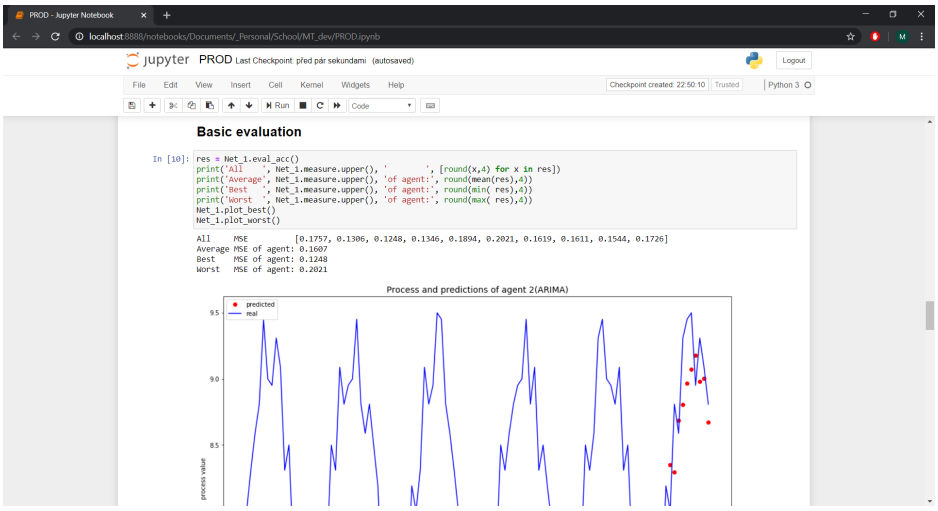

Figure 4.1: Jupyter – illustration of the environment.

Figure 4.1 is a screenshot from Jupyter notebook and shows Jupyter environment, running in the browser. The combination of code cells and markdown cells makes it a useful tool for visualization and presentation of the data and outputs of data analysis, especially if it was processed by Python libraries.

4.1.3 Anaconda

Anaconda is a platform distributing Python packages, in its basic form freely available even for commercial use. Anaconda makes it easier to run Jupyter along with other packages, especially on some platforms, such as Microsoft Windows OS.

Anaconda was used in version 2020.02.

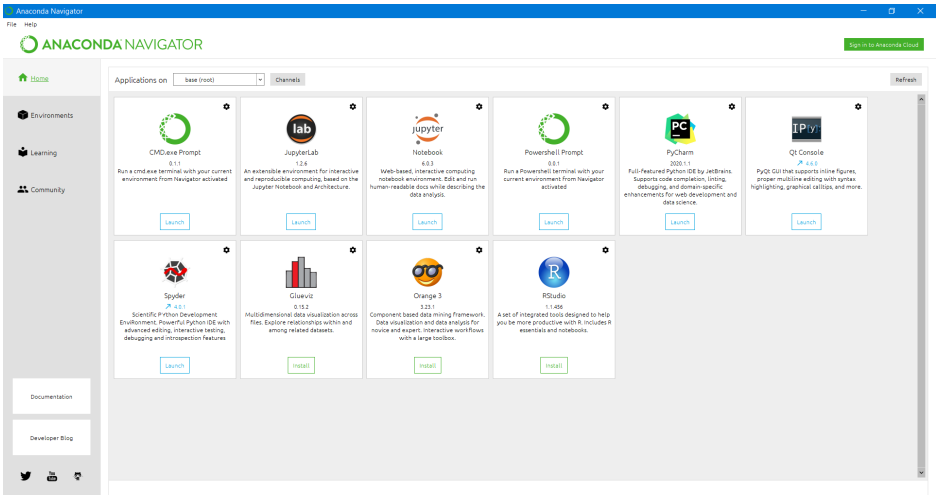

Figure 4.2: Anaconda – illustration of the environment.

Figure 4.2 shows the environment of Anaconda Navigator, from which Jupyter can be started. Furthermore, Anaconda Prompt can be used to install packages, using *conda* or *pip* commands.

4.2 Module implementation – structure

In order to demonstrate the variants described in Chapter 3, a Python implementation was developed. Individual parts are to be found in the module *model_fusion.py*.

Mentioned module *model_fusion.py* is comprised of four classes used to implement the agent communication, predictions, time management, evaluation, and overall workflow:

- CProcess
- CModel
- CAgent
- CNetwork

Constants for limiting maximum size (and thus maximum running time) of the inner variables of mentioned classes are also defined in this file. These inner variables are used because of better memory management, faster access than standard Python lists, and for better readability of the code, too.

4.2.1 CProcess

Class CProcess is a wrapper to provide other objects (or programmer) with clear access to the value of the process in given time. Both real value (used for accuracy evaluation after running the experiment) and observed value (for the purpose of individual agents' observations) can be accessed using multiple methods, and in the desired time period. Illustrative example of extracting the range of values (without documentation):

```
def val_range(self, end, start=0, agent_id = -1):
    if agent_id == -1:
        return self.values[start:end]
    return self.observed[agent_id][start:end]
```

4.2.2 CModel

Class CModel is a wrapper to provide agents with unified access to fitting the model, making the predictions, and possibly getting certainties regarding the prediction from the models, if provided (enabled by the model and implemented).

Individual models are hardcoded in this class and differentiated by their id obtained by a static method:

```
def translate_type(model_name):
    return CModel.types.index(model_name)
```

Each type of model handles the prediction separately. Method *refit()* is used to fit the model based on the data provided as argument when called. Then, the method *get_prediction()* can be called to obtain the model prediction for given time. Usually, only the prediction for the next time step is implemented in this class and therefore makes sense. If the model supports calculation of the certainty of its prediction, it is determined while calling the method *get_prediction()* and its current value can be accessed via method *get_certainty()*. Certainty can be any number, but the simple fact should apply – the higher the model is self-confident about the quality of the current prediction, the higher the certainty.

This manner of handling multiple model types is not ideal, but is rather simply extensible if new types of models are to be added and is sufficient for that matter. Individual capabilities of the models are mentioned further in the text.

4.2.3 CAgent

Class CAgent implements the communication between the inner model (instance of the CModel class, assigned by *assign_model()* method) and the outer world (other models/agents and process, represented by a CNetwork object and its methods).

Process observing

The method *observe_process()* stores the argument provided as the observed value of the process. If the observations of the neighbours are to be comprised, calling the *share_observation_update()* method with the neighbours' observations as argument updates the observed value to the arithmetic mean of the agent's neighbours, agent itself included.

Sharing the info

The methods *share_prediction_get()* and *share_cert_get()* are used to communicate the neighbours' predictions and certainties to the given agent.

Making the prediction

Method *fit_ipredict()* is a wrapper to call the agent's model's functions to fit and then make an inner prediction. To be specific:

```
def fit_ipredict(self):
    self.imodel.refit(self.observed[:self.time])
    self.ipred = self.imodel.get_prediction(self.time)
    return self.ipred
```

Method *make_prediction()* is the key method – it provides the functionality of incorporating other agents' predictions and based on the cooperation mode (i.e. variant proposed in Chapter 3), makes the final prediction of the agent. Pivotal part of the code, edited for demonstration purposes, looks like this:

```
shifted_w   = [x*y for x,y in zip(
                weights, neighbour_certs)]
shifted_w_n = [x/sum(shifted_w) for x in shifted_w]

if   mode == 'average':
    last_pred  = mean(neighbour_preds)
elif mode == 'w_average':
    last_pred  = np.dot(neighbour_preds, shifted_w_n)
elif mode == 'w_median':
    last_pred  = weighted_median(neighbour_preds,
                                 shifted_w_n)
elif mode == 'ignore':
```

```
    last_pred  = self.ipred
elif mode == 'best_neighbour':
    bst_nbr_id = shifted_w_n.index(max(shifted_w_n))
    last_pred  = self.neighbour_preds[bst_nbr_id]
```

The list variables *weights*, *neighbour_certs*, and *neighbour_preds* store also the values of the agent itself (in the last position). The variable *shifted_w* stores weights of the neighbouring agents adjusted by the certainty provided by them. Those new weights have to be normalized (variable *shifted_w_n*). The *last_pred* variable is set to the currently valid prediction made.

Calculating neighbours' weights

Method *update_neighbours_params()* sets the weights based on the given measure, default is MSE. To avoid storing all historical data of the other agents (i.e. past errors) locally, only the mean value is stored and the weights are updated based on the number of predictions made (thus, the result is the same). It is calculated from the errors based on their predictions and observed value of the process like this:

```
def update_neighbours_params(self, measure='mse'):
    time_weight = time - train_time
    if   measure == 'mse':
        cur_e = [abs(observed[time]-x)**2
                 for x in neighbour_preds]
    elif measure == 'mae':
        cur_e = [abs(observed[time]-x)
                 for x in neighbour_preds]
    new_e = [ x*(1           /(time_weight+1))
            + y*(time_weight/(time_weight+1))
              for x,y in zip(cur_e, avg_e)]
    avg_e   = new_e
    weights = CAgent.get_weights(new_e)
```

The variable *avg_e* stores the average error and is updated after the recalculation and incorporating the current prediction error. After that, weights are updated. To do so, the following static method is called:

```
def get_weights(errors):
    ws   = [1 - x/sum(errors) for x in errors]
    ws_n = [x/sum(ws) for x in ws]
    return ws_n
```

Because the certainties have to be mapped from "random" values provided by the models into the reasonable numbers, preferably close to 1, the static method *binner()* is implemented. Based on the distribution (the quantiles) of

the array, the value is mapped into one of the values of the certainty mask (usually, that is something like a triplet [0.95, 1.00, 1.05]).

4.2.4 CNetwork

Class CNetwork implements the flow of the time and ensures the information transfer between agents. It also collects the predcited values for further analysis. Several auxiliary methods are also created to simplify the code generation necessary and depiction of the predicted values.

The method *make_time_steps()* calls the method *make_time_step()* for given number of times, records the progression of the run, and measures the execution time.

The method *make_time_step()* executes one round of prediction – all agents fit their inner model, predict the next value, and then share the predicted value with their neighbours (the appropriate *CAgent* methods are called with the corresponding arguments). After all the agents get the predictions of neighbouring nodes, they make the final prediction based on the cooperation mode specified when the network was created. Finally, the agents observe the process, potentially exchange the observed values (if specified by the network) and update the parameters denoting weights of the neighbours' predictions.

The method *eval_acc()* is used for evaluation of the prediction accuracy.

4.2.5 Models used

This section describes various models and their possible functionalities supported by the class CModel. The prediction certainty is implemented in all models – Prophet uses the ratio of size of the prediction interval divided by the prediction value (or rather 1 minus this value, because the larger the interval, the lower the certainty). The other models use the negative BIC.

AR

AR uses autoregressive model (AutoReg from statsmodels module). The *lags* keyword argument is supported by the CModel wrapper.

ARIMA

ARIMA uses autoregressive integrated moving average model (ARIMA from statsmodels module). The *order* keyword argument is supported by the CModel wrapper.

SARIMAX

SARIMAX uses seasonal autoregressive integrated moving average model (SARIMAX from statsmodels module). The *order* keyword argument and *sea-*

sonal_order keyword arguments are supported by the CModel wrapper.

Prophet

Prophet uses Prophet model (from fbprophet module). The *start* keyword argument is supported by the CModel wrapper.

Exponential Smoothing

ExpSm uses exponetial smoothing model (ExponentialSmoothing from statsmodels module). The *seasonal_periods* keyword argument is supported by the CModel wrapper. The *window* keyword argument is implemented – only last *window* observations are used to fit the model.

4.2.6 Code documentation

Because not every single detail can be discussed, in order to help with code orientation and usage, simple documentation was generated.

Created code is briefly commented with docstrings and using *reStructuredText* format. Simple documentation was generated by pydoc. This generated HTML document is available on the enclosed DVD.

Figure 4.3: Excerpt from simple HTML documentation generated by pydoc.

Figure 4.3 shows the structure of generated documentation. However, methods are commented using *reStructuredText* as markup language, so that more solid documentation (e.g. using Sphinx) can be generated and maintained in case of further work on the code or incorporation in another project.

4.3 How to run the implementation

This section describes how the code can be run and results visualized using the created Jupyter notebook(s), available on enclosed DVD.

4.3.1 Environment

For visual and demonstrative purposes, a Jupyter notebook was made. It can be launched either from command line, or using the Anaconda Navigator. Some Python packages have to be installed, possibly in specific version – *conda* or *pip* commands are able to do that. More details are in the first part of this chapter.

4.3.2 Jupyter notebook overview

For convenience, the notebook is recommended to be run at once using the *Restart & Run All* option in the *Kernel* menu, as shown in Figure 4.4.

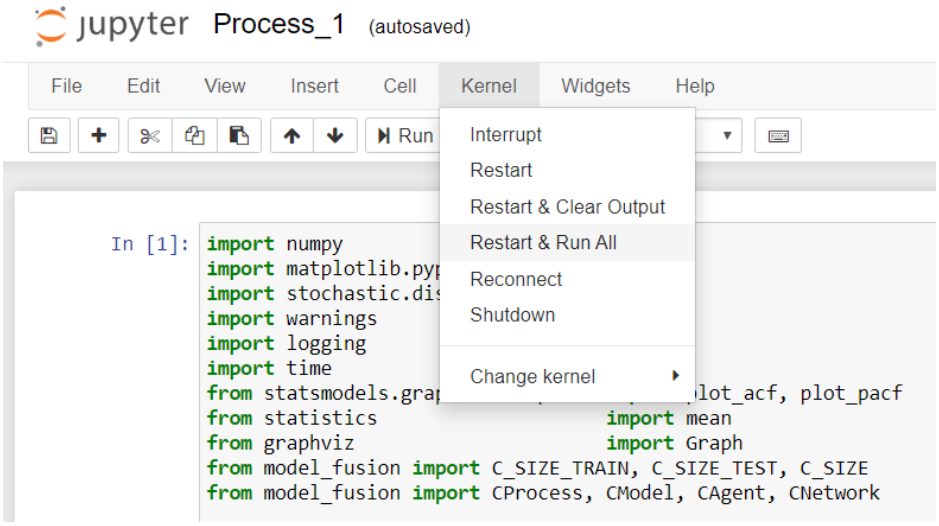

Figure 4.4: Simple running of Jupyter notebook.

First cell of the notebook contains necessary imports. Several of them are from the implemented *model_fusion* module. Therefore, the file *model_fusion.py* should be in the same folder as the notebook in order to be imported. Several warnings or informative messages from *fbprophet* and other libraries are often displayed when running some parts of the code. The behaviour can be easily managed by (un)commenting the function calls in the first cell.

In the next cells, the process is created, along with the noise for the observers. Also, several process statistics are shown.

Creating the objects based on implementation follows. Firstly, the models are specified. Then, the agents and network structure – to ease the code creation, the function *CNetwork.generate_code()* was created. It prints the code, which can be then copied, modified as desired, and finally run. Excerpt from this part is illustrated in Figure 4.5.

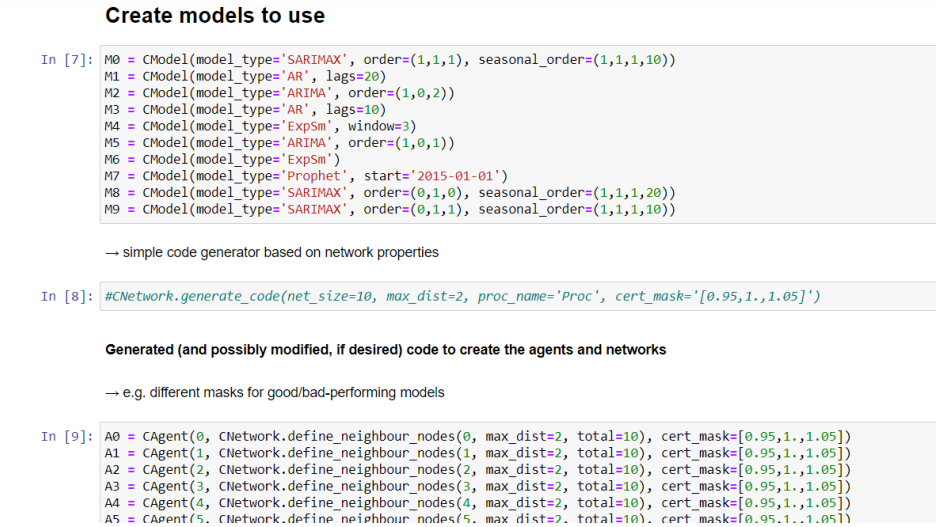

Figure 4.5: Models creation.

The next section runs the experiment for desired number of steps. The default value is set in the developed module to 100. In case of running the experiment for more steps, the variable should be changed in the module because the created objects depend on it.

After that, the predictions of individual agents are plotted and basic statistics shown.

Finally, agent details – e.g. individual certainties – can be examined, but the manual selection of what is a matter of interest is recommended. At the end of the notebook, the network structure is displayed.

4.4 Code discussion and further work on the implementation

Current implementation is sufficient for making various experiments related to the network structure, node degrees, and many more. However, its primary usage is to simulate the network communication, and could not be put into real-life production usage that easily as such. The exchange of information between agents is provided by a centralized unit, instance of CNetwork class. Multiple changes would have to be done, mainly regarding the time synchronization. The advantage of doing so would be parallel and possibly

independent execution of the code, mainly of model fitting. The slight disadvantage would be less managable running of experiments and data gathering.

Subsequent development could be focused on the predictive capabilities in the sense of incorporating more types of models. Certainties calculations could be also tweaked or corrected in order to offer more robustness and reliability.

Chapter 5

Experiments and evaluation

This chapter deals with hands-on experience with the implemented variants. Demonstrations of several configurations are made in order to empirically evaluate the quality of individual aspects of the variants. Finally, after the evaluation of executed experiments, some other methods and approaches that could be tested are proposed.

5.1 General

This part aims to compare the proposed variants and to put various configurations of the agents cooperation to the test.

5.1.1 Properties examined

The most important attributes explored are:

- variant used to combine the individual predictions – can be one of the five proposed in Chapter 3: *ignore*, *best_neighbour*, *average*, *weighted_average*, *weighted_median*
- number of neighbouring agents (node degree) – 2, 4, 6, or 8

Other attributes investigated and compared include:

- certainty mask used – for simplicity, mask is same for all of the agents, usually [0.95, 1.0, 1.05], but is varied in some cases
- inner metric used to evaluate the prediction quality – usually MSE, MAE is tested in some cases
- sharing the observed value of the process – the observed value of the process is usually shared, but not in some cases
- network size – in most cases, the network has 10 agents, but larger network is also considered

5.1.2 Comparing the configurations

Generally, in each configuration of the experiment, MSE is used as metric to evaluate – and more importantly, compare – the ability of the network to predict the future values. To be specific:

Average agent performace (MSE) arithmetic mean of the individual MSE values of each agent; primary indicator used to compare the configurations

Best agent performace (MSE) MSE of the best predciting agent; in comparison, this indicates if the network enables to preserve or even enhance the predictive capabilities of the well performing agents

Worst agent performace (MSE) MSE of the worst predciting agent; this indicates if the network enables to enhance the inferior models

To run each configuration, 100 observations of train data (observed values of the process known to the agent) was provided to the agents and then, a round of 100 iterative predictions (and refitting of the models) was run in the Jupyter notebook.

5.1.3 Network topology

The network structure is similar in all experiments. Every time, it is a regular graph – every node has the same degree. Also, it is constructed as n-hops – resulting in a circular structure, as shown in Figure 5.1.

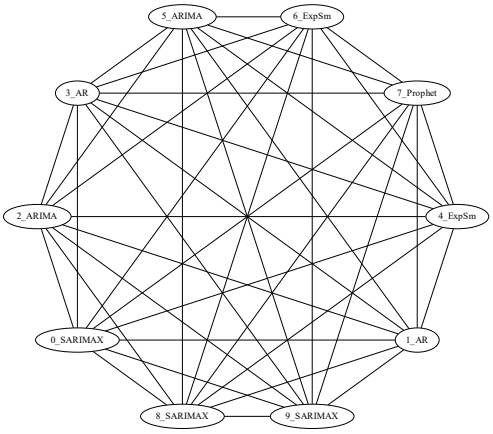

Figure 5.1: Network topology – 10 agents, each with 8 neighbours.

In two cases, larger networks are created in attempt to improve the prediction accuracy. Their structures are depicted in Figures 5.2 and 5.3.

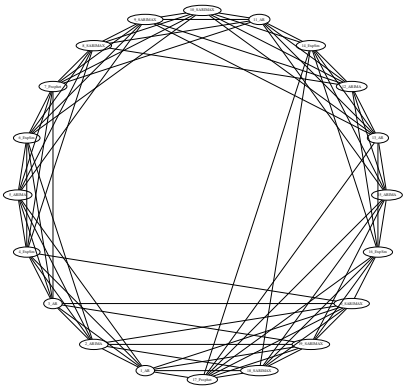

Figure 5.2: Network consisting of 20 agents, each with 8 neighbours.

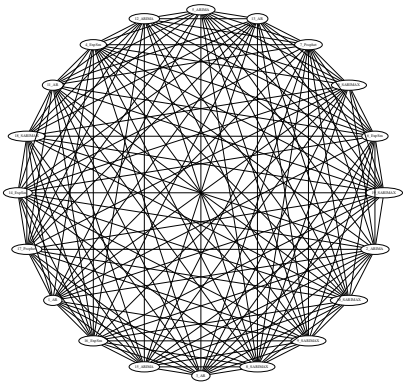

Figure 5.3: Network consisting of 20 agents, each with 16 neighbours.

5.1.4 Models and their configuration used in the network

Following models and their configurations were used to model and predict the value of the stochastic process:

Agent 0 used *SARIMAX* with *order=(1,1,1), seasonal_order=(1,1,1,10)*

Agent 1 used *AR* with *lags=20*

Agent 2 used *ARIMA* with *order=(1,0,2)*

Agent 3 used *AR* with *lags=10*

Agent 4 used *ExpSm* with *window=3*

Agent 5 used *ARIMA* with *order=(1,0,1)*

Agent 6 used *ExpSm*

Agent 7 used *Prophet* with dummy *start='2015-01-01'*

Agent 8 used *SARIMAX* with *order=(0,1,0), seasonal_order=(1,1,1,20)*

Agent 9 used *SARIMAX* with *order=(0,1,1), seasonal_order=(1,1,1,10)*

5.2 Process 1

Following lines are dedicated to the first process on which the proposed variants are tested and compared.

5.2.1 Test data

First process used for modelling is generated artificially. It is a sum of sine function (with period 20) and Bernoulli process *Bern(p=0.6)*.

This process was chosen mainly to guarantee both periodicity and random elements in the data.

Figure 5.4 shows the real values of the process. Its mean value is 8.193 and standard deviation is 0.753.

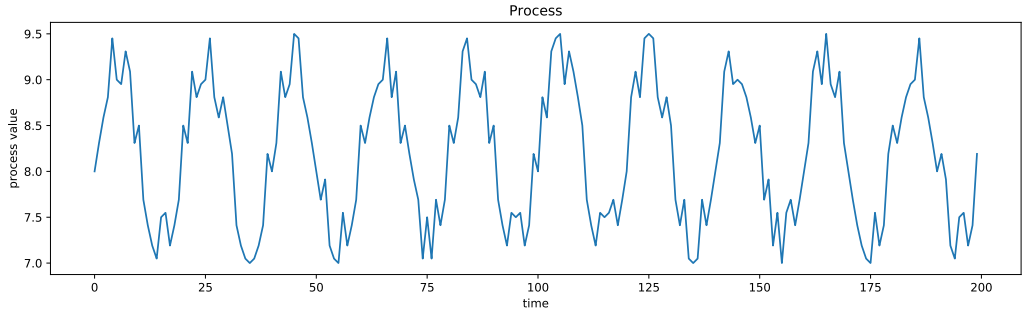

Figure 5.4: Plot of the process 1.

It should be noted that some of the models are better because of their a priori coefficients – e.g. for agent 1, the number of lags is equal to the period, so this model (and possibly its neighbours) is one of the candidates to generally be the best model in this network.

Each agent is provided with slightly different observed values based on random, gaussian noise $\mathcal{N}(0, 0.01^2)$. This is shown in Figure 5.5. The individual differences are very tiny, but can be spotted by eye only with some amount of effort made.

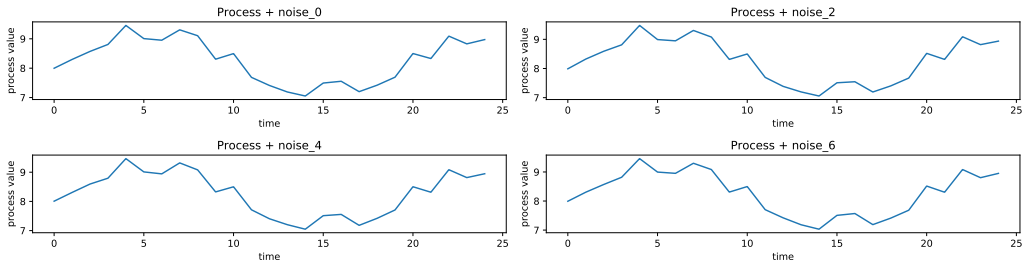

Figure 5.5: Detail on several selected plots of observed values of process 1.

As we can see in Figure 5.6, ACF and PACF behave as could be expected based on the process properties, mainly periodicity.

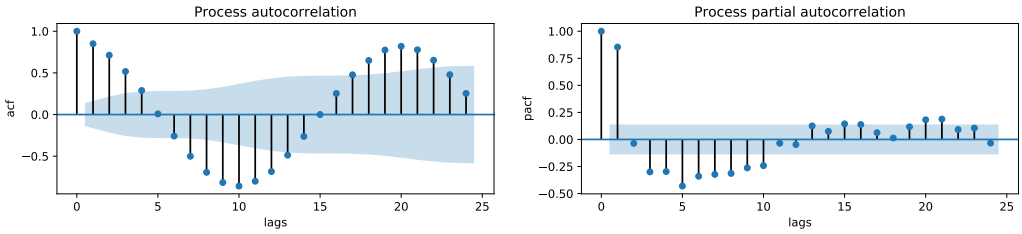

Figure 5.6: ACF and PACF of process 1.

5.3 Results – process 1

Following part describes and examines all the experiments in setting and configuration made with the data from process 1. Several tables with results are presented for clarity – values in green signify the best value (lowest MSE) in the row of the given table (specific configurations), values in red denote the worst value in the row and values in grey signify that the value is taken as reference from previous experiments, if not stated specifically otherwise.

5.3.1 Comparison of variants with differing number of neighbours

As one of the main points is to compare the various proposed methods and study the behaviour of the network with different number of neighbours, different number of neighbours is tried out for each variant. Other attributes stay the same.

For four neighbours, MAE is tested as the inner measure to evaluate the errors of past predictions in order to calculate weights of the neighbours.

4 neighbours

Table 5.1: Process 1 – comparison of results for 4 neighbours.

Nodes	10	10	10	10	10
Neighbours	4	4	4	4	4
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	0.1803	0.1150	0.1085	0.1025	0.1170
Best agent performance (MSE)	0.0807	0.0823	0.0781	0.0770	0.0824
Worst agent performance (MSE)	0.5931	0.1724	0.1467	0.1419	0.1512

Given that each agent has 4 neighbours, weighted average variant gives the best results in all of the three from average, best, and worst agent performances. Details are listed in Table 5.1.

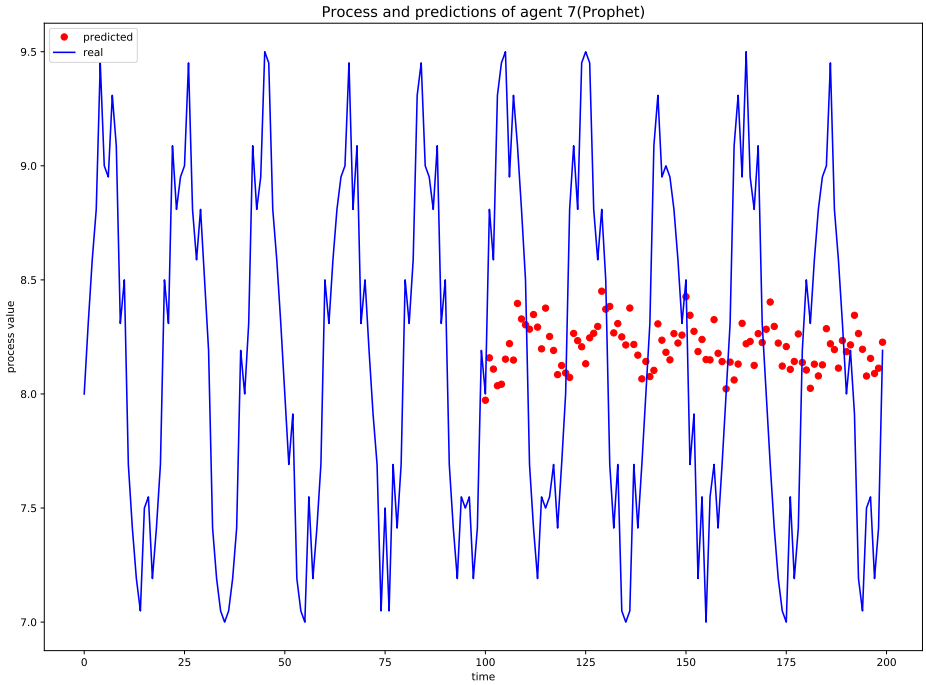

Figure 5.7: Predictions of the worst performing agent in variant 0 – *ignore*.

Figure 5.7 displays the predictions of the worst agent 7 (Prophet) in variant 0 (ignore), that is the configuration from the first column of Table 5.1. Its

MSE reaches almost 0.6, but every other variant lowers this extreme value. Besides, in each of the other variants, the worst performing agents is either agent 6 or agent 8 (neighbours of agent 7), but not the agent 7 itself.

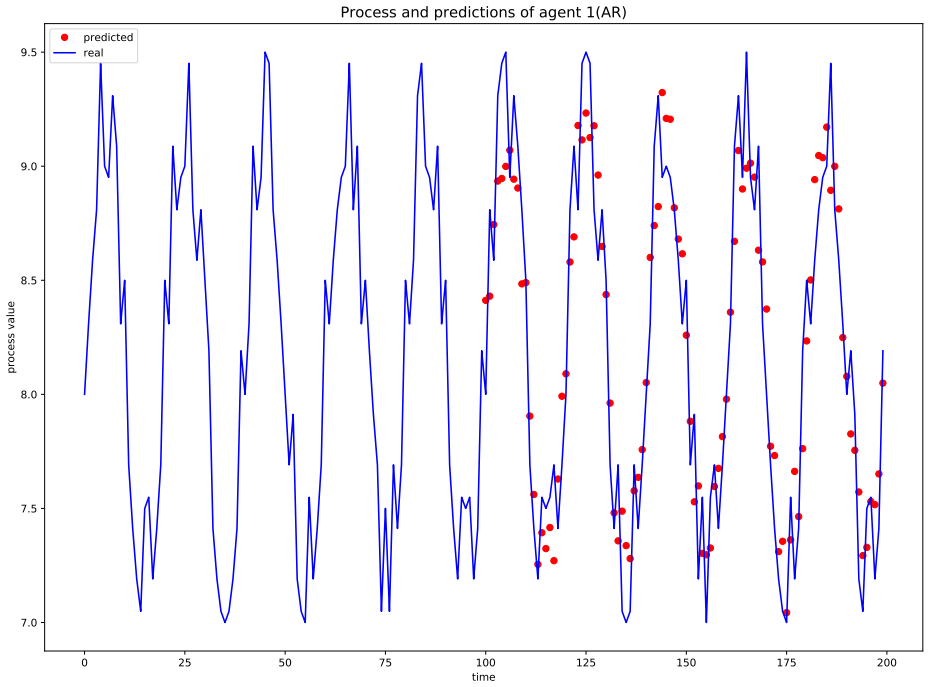

Figure 5.8: Predictions of the agent with lowest MSE, across all variants and modifications.

Figure 5.8 shows the predictions of agent 1, that is AR model with lags up to 20 from weighted_average variant. This is the configuration from the fourth column of Table 5.1. It performs the best from all of the tested variants and modifications with its MSE equal to 0.0770.

Table 5.2: Process 1 – comparison of results for 4 neighbours, MAE.

Nodes	10	10	10	10	10
Neighbours	4	4	4	4	4
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MAE	MAE	MAE	MAE	MAE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	0.1803	0.1148	0.1085	0.1054	0.1170
Best agent performance (MSE)	0.0807	0.0823	0.0781	0.0775	0.0824
Worst agent performance (MSE)	0.5931	0.1740	0.1467	0.1444	0.1512

Table 5.2 presents the results with MAE used as inner metric to evaluate the prediction qualitities of agent and its neighbours and calculate weights of the neighbouring predictions. Because variants *ignore* and *average* do not do these calculations, their results were, not surprisingly, equal to the results displayed in Table 5.1. Therefore, these two variants are depicted in grey colour.

When compared with MSE variant mentioned before, both the best and the worst agents in the MAE variant were equal or worse. In the *best_neighbour* variant, the average performance of MAE variant was slightly better, but that is the only case.

2 neighbours

Table 5.3: Process 1 – comparison of results for 2 neighbours.

Nodes	10	10	10	10	10
Neighbours	2	2	2	2	2
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	0.1804	0.1148	0.1228	0.1114	0.1228
Best agent performance (MSE)	0.0809	0.0841	0.0862	0.0819	0.0829
Worst agent performance (MSE)	0.5953	0.1677	0.1960	0.1665	0.1620

As visible in Table 5.3, in the case of only 2 neighbouring agents, the results are generally worse than in the case of each agent having four neighbours.

6 neighbours

Table 5.4: Process 1 – comparison of results for 6 neighbours.

Nodes	10	10	10	10	10
Neighbours	6	6	6	6	6
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	0.1807	0.1033	0.1021	0.0974	0.1083
Best agent performance (MSE)	0.0806	0.0864	0.0809	0.0800	0.0822
Worst agent performance (MSE)	0.5982	0.1724	0.1293	0.1242	0.1396

Table 5.4 provides comparison of the five variants used with each agent sharing predictions with 6 other nodes. The average performace of agents gets better than in the case of 4 neighbours scenario, with the exception of variant *ignore*.

8 neighbours

Table 5.5: Process 1 – comparison of results for 8 neighbours.

Nodes	10	10	10	10	10
Neighbours	8	8	8	8	8
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	0.1804	0.0960	0.0982	0.0948	0.1032
Best agent performance (MSE)	0.0807	0.0921	0.0891	0.0882	0.0846
Worst agent performance (MSE)	0.5970	0.1044	0.1044	0.1007	0.1197

As visible in Table 5.5, the best performing variant (regarding the average prediction of all agents in the network) across all configurations using certainties of the prediction is *weighted_average* combined with 8 neighbouring nodes.

Also, the network structure is shown in Figure 5.1 mentioned earlier.

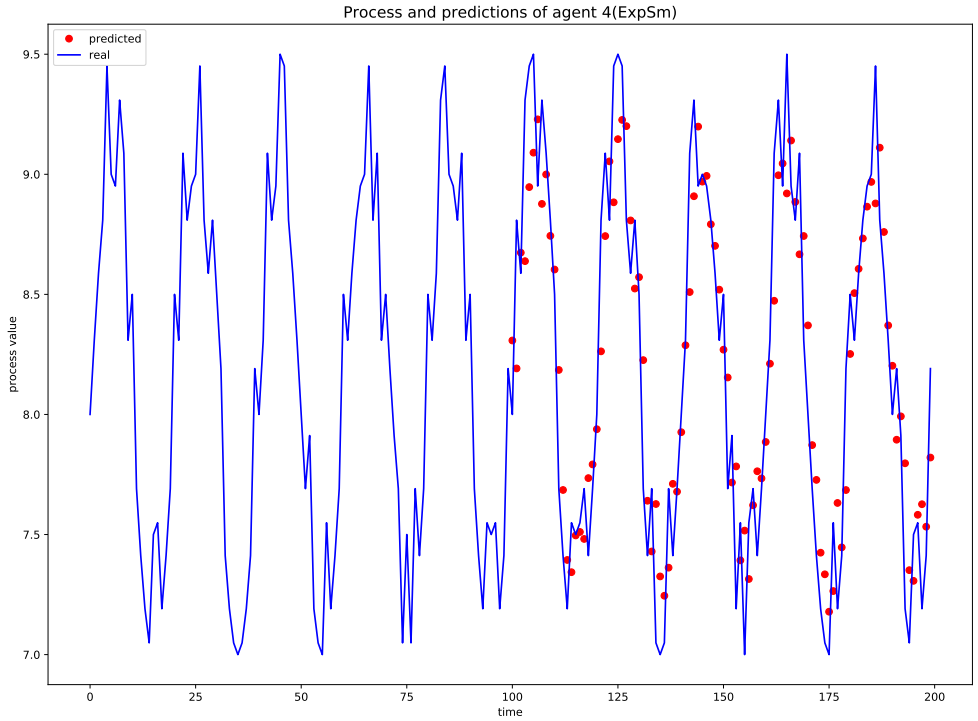

Figure 5.9: Predictions of the worst agent in the best performing variant, compared across variants in which certainty was used.

Figure 5.9 shows the predictions of agent 4 (Exponential smoothing, with window equal to 3 – only last three predictions are used) in the configuration of 8 neighbours in the *weighted_average* variant. This configuration produced the best agents' average MSE (0.0948), and even its worst agent predicts better than 8 out of 10 agents in the variant *ignore*, with any number of neighbours (individual agent predictions can be reviewed in any of the executed Jupyter notebooks).

5.3.2 Variating other attributes

After performing and evaluating basic configurations of the network with different number of neighbours, two interesting configurations were selected and further investigation was made.

Variating parameters of the best configuration

Variation *weighted_average* with eight neighbouring nodes – fourth column from Table 5.5 – was chosen and furthermore referred to as the "best" configuration because of its lowest average MSE. Several experiments were made

in order to try and find a configuration similar to this one that would give at least slightly better results.

In the first round, these are:

- larger network – 20 nodes of degree 16 (doubled) were used, as can be noticed in the diagram 5.3; the other ten nodes use same models as the first ten nodes, but they do observe different values of the process
- not sharing the observed value of the process
- AIC used as the criterion when calculating certainty

Table 5.6: Process 1 – comparison of various modifications of the "best" configuration.

Nodes	10	20	10	10
Neighbours	8	16	8	8
Cooperation	w_average	w_average	w_average	w_average
Measure	MSE	MSE	MSE	MSE
Shared observation	True	True	False	True
Note on change	"best" configuration	larger network	observations not shared	AIC as certainties
Average agent performance (MSE)	0.0948	0.0961	0.0948	0.0949
Best agent performance (MSE)	0.0882	0.0874	0.0883	0.0883
Worst agent performance (MSE)	0.1007	0.1061	0.1008	0.1008

Table 5.6 shows the results of the attempts to improve already well performing configuration. The bigger network configuration (second column with the data in the mentioned table) found a better performing best agent, though at the cost of worsening in the remaining two compared categories, average and worst MSE. The other two experiments, not sharing the observed values and using AIC, give almost the same numbers as the original "best" configuration.

In the second round, extent of certainties of the models was changed in order to investigate the impact:

- larger span in certainty masks of all agents – namely [0.85, 1, 1.15] instead of default [0.95, 1 and 1.05]
- smaller span in certainty masks of all agents – namely [0.98, 1, 1.02]
- combined span in certainty masks – agents A3, A4, A5, A6, and A8 performed worse than the other agents and therefore get lower span masks [0.98, 1, 1.02]; remaining agents get larger span of certainty mask [0.85, 1, 1.15]

- inversed combined span in certainty masks – same as the mixed span, but inversed, i.e. the worst agents A3, A4, A5, A6, and A8 get larger span of certainty mask [0.85, 1, 1.15] and the rest gets lower span masks [0.98, 1, 1.02]

Table 5.7: Process 1 – comparison of certainties modifications of the "best" configuration.

Nodes	10	10	10	10	10
Neighbours	8	8	8	8	8
Cooperation	w_avg	w_avg	w_avg	w_avg	w_avg
Measure	MSE	MSE	MSE	MSE	MSE
Shared observ.	True	True	True	True	True
Note on change	"best" config.	larger mask	smaller mask	comb_1 mask	comb_2 mask
Average agent perf. (MSE)	0.0948	0.0954	0.0946	0.0952	0.0949
Best agent perf. (MSE)	0.0882	0.0886	0.0881	0.0883	0.0884
Worst agent perf. (MSE)	0.1007	0.1022	0.1007	0.1020	0.1010

The results provided by Table 5.7 indicate that the changes have rather small effect and it seems that the smaller the certainty span, the better.

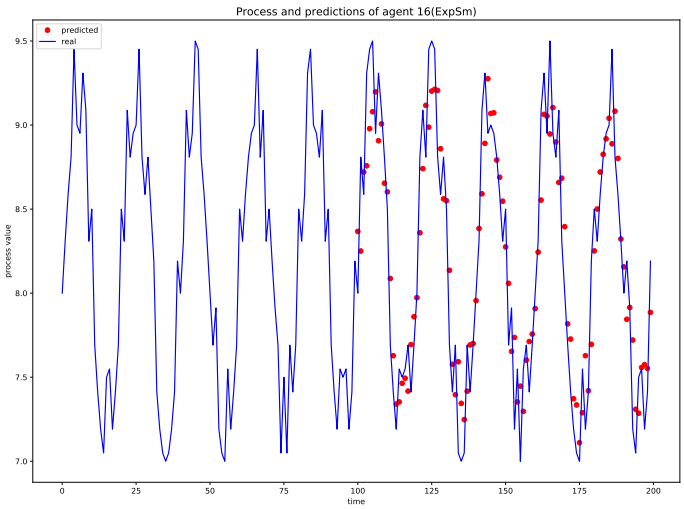

Figure 5.10: Predictions of the best agent from the modifications of the "best" performing variant.

Figure 5.10 shows the predictions of the agent 16 (Exponential smoothing) with MSE equal to 0.0874, the best model found while tweaking the "best" configuration. The agent itself is better than the originally "best" agent in this configuration, but still far behind the totally best agent (across all configurations) with its MSE as low as 0.0770.

Varying parameters of the worst configuration

Variation *weighted_median* with four neighbouring nodes – fifth column from Table 5.1 – was chosen and furthermore referred to as the "worst" configuration. It was chosen as such, even though it is not actually the worst one. The point is to select a configuration employing either *weighted_average* or *weighted_median* variant and with each node having degree at least 4, and from these, it is the worst one. Several experiments were made in order to try and find a configuration that would perform better than this "worst" configuration.

Very similarly as before, the tested changes are:

- larger network – 20 nodes of degree 8 (doubled) used, as can be noticed in the diagram 5.2; the other ten nodes use same models as the first ten nodes, but they do observe different values of the process
- not sharing the observed value of the process
- AIC used as the criterion when calculating certainty

Table 5.8: Process 1 – comparison of various modifications of the "worst" configuration.

Nodes	10	20	10	10
Neighbours	4	8	4	4
Cooperation	w_median	w_median	w_median	w_median
Measure	MSE	MSE	MSE	MSE
Shared observation	True	True	False	True
Note on change	"worst" configuration	larger network	observations not shared	AIC as certainties
Average agent performance (MSE)	0.1170	0.1035	0.1163	0.1170
Best agent performance (MSE)	0.0824	0.0844	0.0820	0.0824
Worst agent performance (MSE)	0.1512	0.1203	0.1504	0.1512

As recorded in Table 5.8, the larger network provides significantly better results on average (and also with its worst agent), but at the cost of having its best agent non-competitive in comparison with the other.

Again, similarly as before (with the exception of changed set of the worst agents) – in the second round, extent of certainties of the models was changed in order to investigate the impact:

- larger span in certainty masks of all agents – namely [0.85, 1, 1.15] instead of default [0.95, 1 and 1.05]
- smaller span in certainty masks of all agents – namely [0.98, 1, 1.02]
- combined span in certainty masks – agents A3, A4, A5, A6, and A7 performed worse than the other agents and therefore get lower span masks [0.98, 1, 1.02]; remaining agents get larger span of certainty mask [0.85, 1, 1.15]
- inversed combined span in certainty masks – same as the mixed span, but inversed, i.e. the worst agents A3, A4, A5, A6, and A7 get larger span of certainty mask [0.85, 1, 1.15] and the rest gets lower span masks [0.98, 1, 1.02]

Table 5.9: Process 1 – comparison of certainties modifications of the "worst" configuration.

Nodes	10	10	10	10	10
Neighbours	4	4	4	4	4
Cooperation	w_med	w_med	w_med	w_med	w_med
Measure	MSE	MSE	MSE	MSE	MSE
Shared observ.	True	True	True	True	True
Note on change	"best" config.	larger mask	smaller mask	comb_1 mask	comb_2 mask
Average agent perf. (MSE)	0.1170	0.1167	0.1170	0.1170	0.1174
Best agent perf. (MSE)	0.0824	0.0831	0.0824	0.0825	0.0824
Worst agent perf. (MSE)	0.1512	0.1574	0.1512	0.1512	0.1527

Results resemble the situation with variating the parameters of the "best" configuration – changes in certainties have small effect on performance and if so, it gets worse – Table 5.9 is filled with the specific numbers.

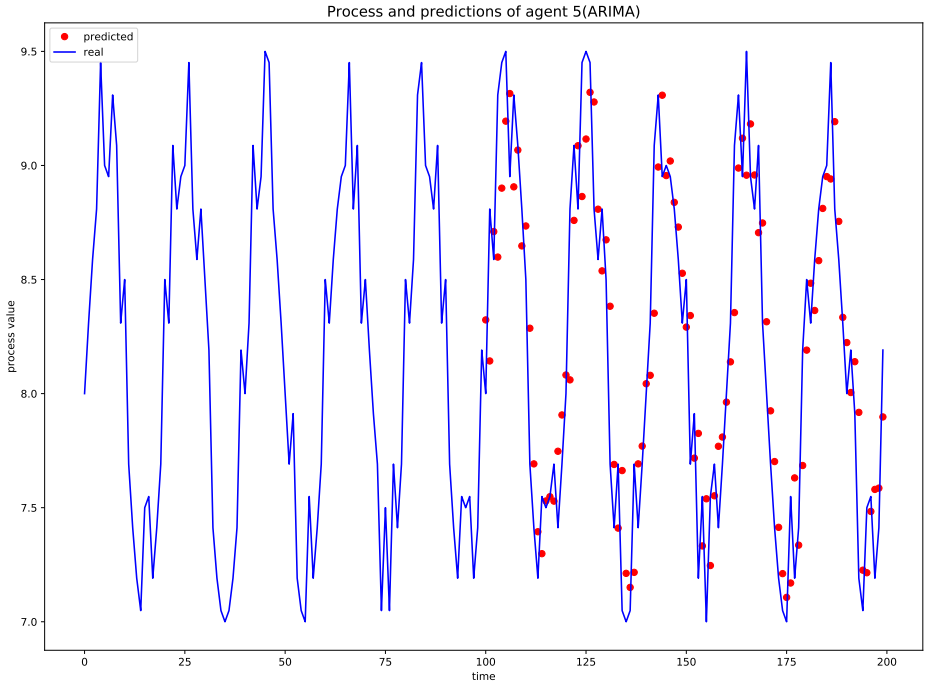

Figure 5.11: Predictions of the worst agent from the best of the modifications of the ”worst” performing variant.

Figure 5.11 depicts the predictions of the worst agent 5 (ARIMA) with MSE equal to 0.1203. Its configuration is in the second column with data of Table 5.8. This configuration is the best from the ”worst” configuration attempted adjustments. It improved the average MSE from 0.117 to 0.1035 – that is relatively high enhancement.

5.3.3 Predictions without using certainty

In following experiments, the configurations do not use certainty (it is always masked to 1). Therefore, *ignore* and *average* variants would have the same results, because they do not incorporate the certainties into their predictions.

Results for variant *weighted_median* were exactly the same as in the case of certainty used (for all of the 2, 4, 6, and 8 neighbours scenarios). That was probably caused by the fact that the weights of the neighbours are usually similar, and the weighted median consistently chose the same values regardless the small changes in the certainty masks. That is why the variant is missing in the following overview of results.

Table 5.10: Process 1 – comparison of results for 2 and 4 neighbours, without certainties used.

Nodes	10	10	10	10
Neighbours	2	2	4	4
Cooperation	best_n	w_average	best_n	w_average
Measure	MSE	MSE	MSE	MSE
Shared observ.	True	True	True	True
Average agent perf. (MSE)	0.1123	0.1115	0.1024	0.1024
Best agent perf. (MSE)	0.0866	0.0824	0.0844	0.0774
Worst agent perf. (MSE)	0.1586	0.1666	0.1557	0.1406

In Table 5.10, colour of the numbers denotes comparison with the same experiment, but run with the certainties equal to [0.95, 1, 1.05] (these values are in Tables 5.3) and 5.1 – green means no certainty performs better, red the other way around. These configurations show that the best models generated perform slightly worse than in the case when certainty is used. However, the mean value of configurations without certainty used is usually better.

Table 5.11: Process 1 – comparison of results for 6 and 8 neighbours, without certainties used.

Nodes	10	10	10	10
Neighbours	6	6	8	8
Cooperation	best_n	w_average	best_n	w_average
Measure	MSE	MSE	MSE	MSE
Shared observ.	True	True	True	True
Average agent perf. (MSE)	0.0983	0.0974	0.1002	0.0945
Best agent perf. (MSE)	0.0843	0.0800	0.0950	0.0881
Worst agent perf. (MSE)	0.1103	0.1243	0.1042	0.1006

In Table 5.11, colour of the numbers denotes comparison with the same experiment, but run with the certainties equal to [0.95, 1, 1.05] (these values are in Tables 5.4) and 5.5 – green means no certainty performs better, red the other way around, grey is for same numbers.

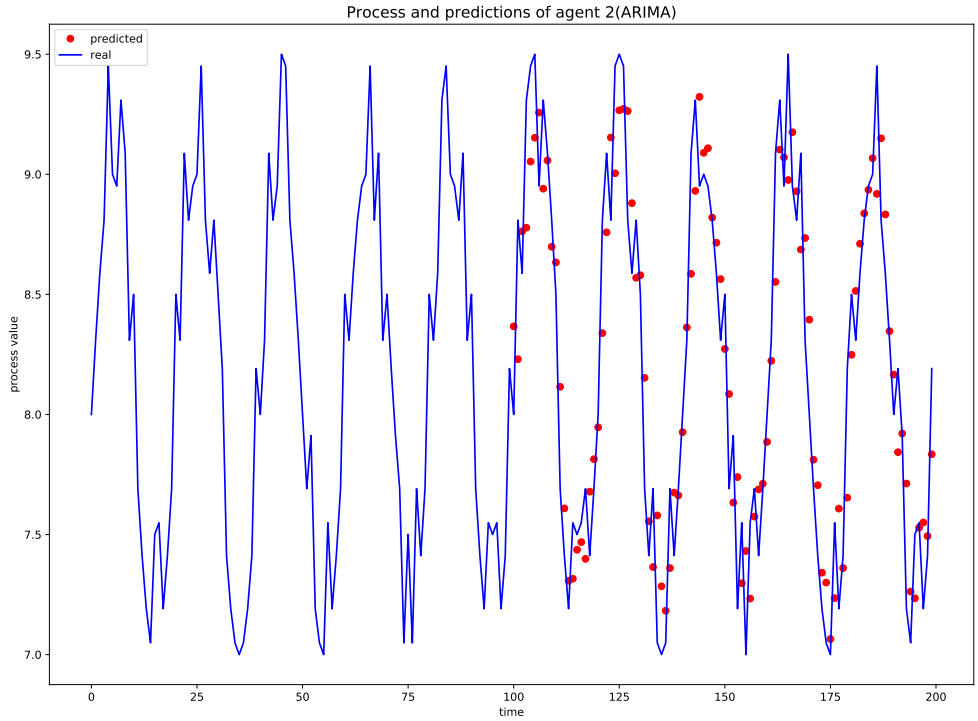

Figure 5.12: Predictions of the best agent in the best performing variant, compared across all variants and modifications.

Figure 5.12 shows the agent 2 (ARIMA) with MSE equal to 0.0881. It is the best agent from the best performing network configuration modelling the process 1 (however, it is not the best agent overall) – the average MSE of the network was only 0.0945. The configuration is the fourth column with data in Table 5.11.

5.4 Process 2

Following lines are dedicated to the second process on which the proposed variants are tested and compared.

5.4.1 Test data

Second process used for modelling is real (not generated) time series. Its values are the daily close values of BTCUSD, from Coinbase exchange. The process start is on 1st January 2018.

The goal of using this data is rather to observe the results behaviour when network configuration is changed. However, the process resembles the random

walk, as the financial data often does, so only some adaptive characteristics of the tested variants can be seen because of the low quality of predictions made by the individual agents.

The dataset was downloaded from `https://www.cryptodatadownload.com/data/coinbase/` and is available on the enclosed DVD.

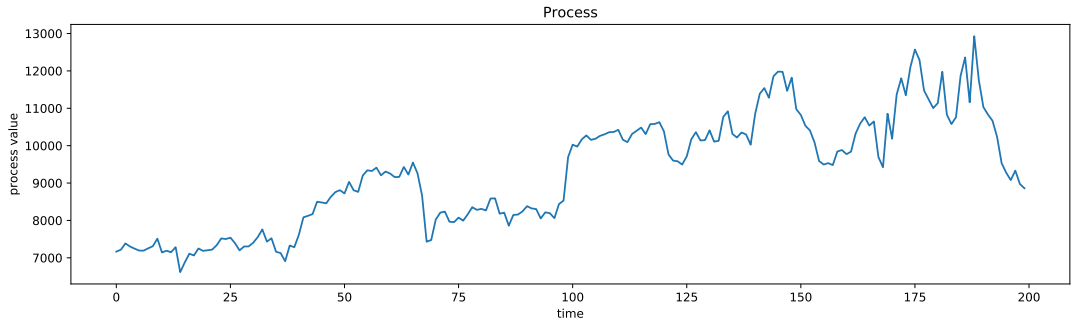

Figure 5.13: Plot of the process 2.

Figure 5.13 shows the real values of the process. Its mean value is 9283.960 and standard deviation is 1494.398.

As in the case of the first process, each agent is provided with slightly different observed values – they are multiplied by a small coefficient based on random, gaussian noise $\mathcal{N}(0, 0.001^2)$. This tries to imitate the situation when multiple sources of data are available, all varying, but in the extent dependent on the value of the process. This is shown in Figure 5.14. The individual differences are tiny, but can be spotted by eye only with some amount of effort made.

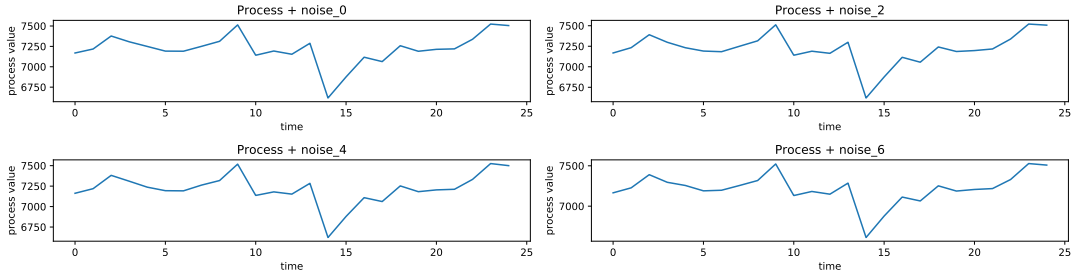

Figure 5.14: Several selected plots of observed values of process 2.

In Figure 5.15, ACF and PACF indicate that the process is a lot like a random walk, dependent mainly on the last value.

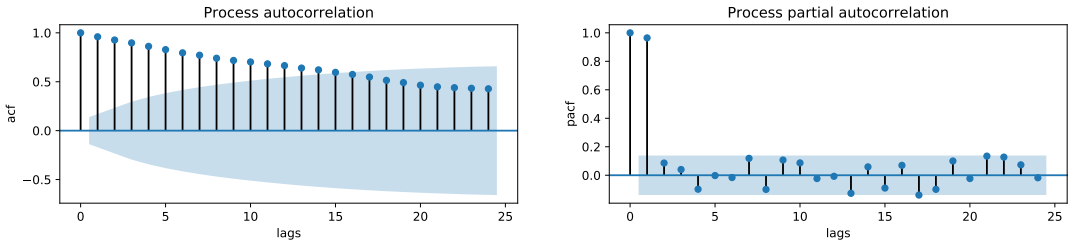

Figure 5.15: ACF and PACF of process 2.

5.4.2 Changes in the models used in the network

In order to at least partially reflect the modelled data, following changes in the model configurations were made:

- Agent 0 used a *SARIMAX* model, with changed *seasonal_order=(1,1,1,7)*
- Agent 7 used a *Prophet* model, with changed *start* – not a dummy variable, but real start date of the time series, i.e. '2018-01-01'
- Agent 8 used a *SARIMAX* model, with changed *seasonal_order=(1,1,1,7)*
- Agent 9 used a *SARIMAX* model, with changed *seasonal_order=(1,1,1,7)*

These changes should help at least a little bit to reflect the real situation, because the previous configuration aimed to model the process with known period.

5.5 Results – process 2

Similarly to process 1, following part describes and examines all the experiments in setting and configuration made with the data from process 2. Several tables with results are presented for clarity – values in green signify the best value (lowest MSE) in the row of the given table (specific configurations), values in red denote the worst value in the row, and values in grey signify that the value is taken as reference from previous experiments

5.5.1 Comparison of variants with differing number of neighbours

Again, as one of the main points of this work is to compare the various proposed methods and study the behaviour of the network with different number of neighbours, different number of neighbours is tried out for each variant. Other attributes stay unchanged.

For four neighbours, MAE is tested as the inner measure to evaluate the errors of past predictions in order to calculate weights of the neighbours.

4 neighbours

Table 5.12: Process 2 – comparison of results for 4 neighbours.

Nodes	10	10	10	10	10
Neighbours	4	4	4	4	4
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	334 700	251 999	255 746	248 407	248 037
Best agent performance (MSE)	240 475	240 929	240 896	238 800	239 860
Worst agent performance (MSE)	1 015 502	266 845	269 583	257 921	255 106

Table 5.12 summarizes results of experiments with configurations where the node degree was equal to 4. Clearly, the weighted variants *weight_average* and *weight_median* outperform the other ones.

Table 5.13: Process 2 – comparison of results for 4 neighbours, MAE.

Nodes	10	10	10	10	10
Neighbours	4	4	4	4	4
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MAE	MAE	MAE	MAE	MAE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	334 700	249 109	255 746	249 969	248 022
Best agent performance (MSE)	240 475	237 130	240 896	240 809	239 932
Worst agent performance (MSE)	1 015 502	264 653	269 583	259 564	255 106

Table 5.13 gives the results with MAE used as inner metric to calculate weights of the neighbouring predictions. Because variants *ignore* and *average* do not do these calculations, their results were, not surprisingly, equal to the results displayed in Table 5.12. Therefore, these two variants are depicted in grey colour.

When compared with MSE scenario tested before (see Table 5.12, variant *weighted_average* has higher MSE in all categories, *weighted_median* is approximately the same and *best_neighbour* performs better in all categories.

2 neighbours

Table 5.14: Process 2 – comparison of results for 2 neighbours.

Nodes	10	10	10	10	10
Neighbours	2	2	2	2	2
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	334 307	254 068	268 307	251 400	248 997
Best agent performance (MSE)	241 113	240 624	235 258	235 456	239 784
Worst agent performance (MSE)	1 016 074	282 961	309 508	269 111	273 371

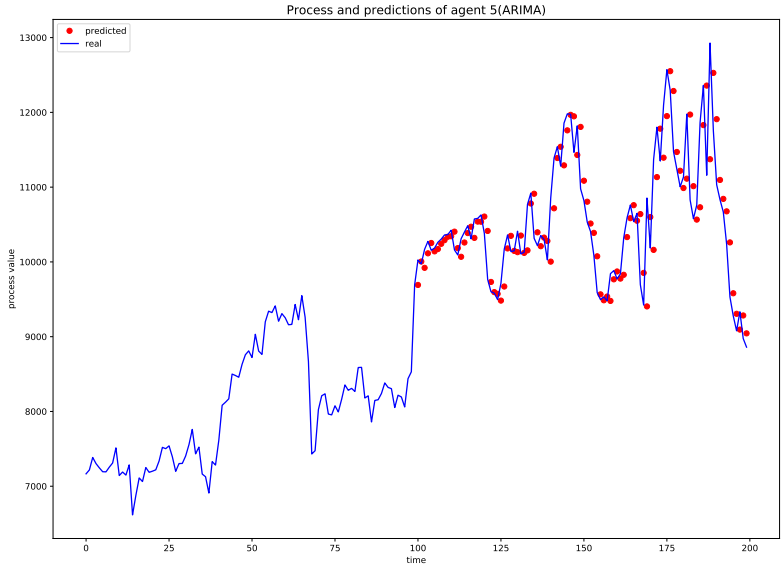

Figure 5.16: Predictions of the best performing agent, all variants considered.

Considering node degree equal to two, Table 5.14 indicates that the average MSE usually gets worse, as expected. Nevertheless, the *average* variant created the best agent predicting the process 2 overall, with MSE as low as 235 258 (third column in the mentioned table) – that is agent 5 (ARIMA). The predictions of this agent are illustrated in Figure 5.16. However, this configuration is also the worst one regarding the average MSE of its agents (if *ignore* variants are omitted).

6 neighbours

Table 5.15: Process 2 – comparison of results for 6 neighbours.

Nodes	10	10	10	10	10
Neighbours	6	6	6	6	6
Cooperation	ignore	best_n	average	w_average	w_median
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	334 681	248 031	250 024	246 397	243 576
Best agent performance (MSE)	240 503	237 623	245 312	239 911	239 183
Worst agent performance (MSE)	1 014 513	265 671	258 146	253 898	246 079

Table 5.15 lists results of configurations in which agents have six neighbours. Each of the individual variants performs in average better than its equivalent with 4 neighbours used.

8 neighbours

Table 5.16: Process 2 – comparison of results for 8 neighbours.

Nodes	10	10	10	10	10
Neighbours	8	8	8	8	8
Cooperation	ignore	best_n	average	w_avg	w_med
Measure	MSE	MSE	MSE	MSE	MSE
Shared observation	True	True	True	True	True
Average agent performance (MSE)	334 314	244 228	246 828	244 854	243 414
Best agent performance (MSE)	240 630	237 628	243 461	241 125	241 776
Worst agent performance (MSE)	1 011 786	251 916	250 198	248 448	244 801

Finally, Table 5.16 presents the results of configurations with 8 neighbours. The best configuration predicting the process 2 is the variant *weighted_median*, with the average MSE equal to 243 414 (fifth column of the table).

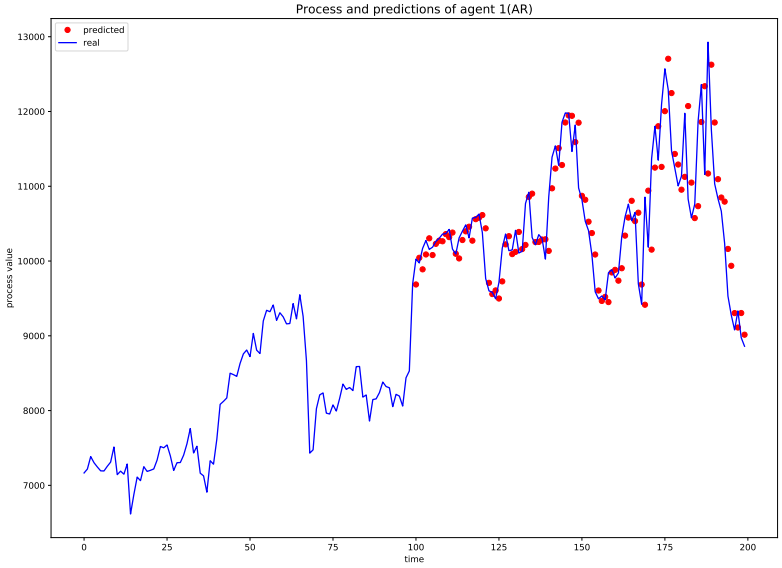

Figure 5.17: Predictions of the worst agent in the best configuration.

Figure 5.17 shows predictions of agent 1 (AR), which is the worst predicting agent in the best configuration in average. Its MSE is 244 801.

5.6 Summary

The lines below sum up the results obtained, compare them, evaluate the methods used, and some possible improvements and further steps are proposed.

5.6.1 Process 1

Many different configurations of agents were tried to predict the value of process 1.

In general, more neighbours means better average prediction capabilities. However, some of the best individual agents emerged from configurations with only 4 neighbours, so based on the experiments made, it should not be universally stated that the higher the node degree the better predictions – it is highly dependent on whether the performance of the best model or of the whole network is preferred.

On comparing the variants used:

- **ignore** – this cornerstone variant performed consistently ignoring the number of neighbours (only neighbours' observed values of the process were used)

- **best_neighbour** – this variant performed better with more neighbours' predictions used, as expected
- **average** – this variant using arithemtic mean of the neighbours' predictions (agent itself included) performed surprisingly well, getting better with higher node degree of agents
- **weighted_average** – this variant performed the best in most cases
- **weighted_median** – this variant performed the second best in most cases

Regarding the other attributes with which some experiments were done:

- **certainty mask** – in many cases, not using certainty or using lower values lead to better results; nevertheless, that is hugely dependent on the quality of the individual models estimates
- **inner metric** – MSE and MAE performed similarly, but few experiments were made with MAE to any conclusion
- **sharing the observed value of the process** – only two configurations were run without sharing the observed value of the process, the results were similar or slightly better
- **network size** – only two configurations were run with larger network, both made slight improvement in one of the studied categories (average MSE and best MSE)

5.6.2 Process 2

Predicting the process 2 was somewhat trickier because of its much more random behaviour. Because of less tests made for this process (any potential results would be highly questionable), it can only be said a few things. First, analogous to process 1, more neighbours means better average prediction. Second, the *weighted_median* variant outperformed all other variants including *weighted_average* with regard to the average MSE of agents in the network. It seems that this variant is a bit more capable of handling more demanding and random data, or rather the lower quality models.

5.6.3 Overall evaluation

To formally answer the questions posed in Chapter 3 – the local predictions can be made better using various techniques of combinations of neighbouring nodes' predictions. This was shown in Chapter 5. Some of the easier methods can, surprisingly, have very good results in several configurations. However, the incorporation of the information from neighbouring nodes using

the more sophisticated weighted variants performs well more consistently and universally.

It must be specifically stated that the variants tested and compared heavily depend on the specific processes they model, besides with the amount of noise associated with the process. More tests would have to be made in order to induce more reliable inferences.

5.7 Adjustments proposed

In further work, following adjustments could be done to study the variants more thoroughly:

- training the models using sliding window
- using the certainty not only for shifting the current prediction, but also for further "(no-)confidence" in the neighbours
- each agent would employ different and **biased** certainty mask given their assumed quality (that means that models would be a priori taken by their neighbours more or less seriously based on the mask)

In the variants *weighted_average* and *weighted_median*, the weights are initialized equally for all the agents in the closed neighbourhood. Nevertheless, if assumptions are made about the quality of the agents' models, the weights could be initialized so that the similar models get lower weights in order to prevent their privileging – similarly as described in [27].

5.8 Alternative approaches

In more abstract perspective, a few more approaches are suggested:

- communication in the network divided in two spheres – firstly in a local cluster, possibly of similar agents, which would agree on the predicted value – and this value would be communicated globally in the network with other clusters
- model switching alteration – each agent would have several of the proposed variants as "strategies" and would change them independently based on their performance

Conclusion

Summary

The breadth and depth of this area is surprisingly eminent. Many ways of modelling and approaches to making predictions are possible and not single one can be universally used.

The way to handle the communication and prediction in the interconnected network of agents was proposed, implemented, and tested. The results made it possible to do at least some evaluation of the proposed variants using both generated and real-world data.

It was demonstrated that using neighbours' predictions can be a good way to locally improve the prediction. Also, it was shown that agent's own certainties of those predictions can lead to some improvements, but at the current state, better results are usually obtained without using the certainty information.

Unsophisticated methods, such as arithmetic mean of predictions, can lead to good results in certain scenarios. However, in most cases, the variants with incorporating past accuracy of the neighbours perform better.

The conducted experiments are easily reproducible and can be run using the implementation on the enclosed DVD.

Prospects

Regarding the presented solution, a lot of experiments could be done with larger networks or with changing parameters of the agents or network. For example, more diversity in the network topology or individual agent's certainty masks can be done right away, with the code as is.

Further work on the implemented solution could be focused on the individual model capabilities – mainly incorporating more types of models (possibly even advanced neural networks like LSTM) and mainly enhancing the way how the models determine the certainty of their predictions.

From higher perspective, there is much unexplored space in experimenting with real-time adaptation of the agent to changes in quality and switching its cooperation strategy. More agent-oriented approach with emphasis on tuning the individual agent's properties and supporting its decision making could lead to much better results in real life.

Bibliography

[1] L. Breiman. Bagging predictors. *Machine Learning*, volume 24, no. 2, Aug. 1996: pp. 123–140.

[2] Y. Freund; R. E. Schapire. A short introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1999, pp. 1401–1406.

[3] J. A. Hoeting; D. Madigan; et al. Bayesian model averaging: A tutorial. *Statistical Science*, volume 14, no. 4, 1999: pp. 382–417.

[4] B. Clarke; B. Yu. Comparing Bayes model averaging and stacking when model approximation error cannot be ignored. In *Journal of Machine Learning Research*, 2003.

[5] K. Dedecius; P. M. Djuric. Sequential estimation and diffusion of information over networks: A Bayesian approach with exponential family of distributions. *IEEE Transactions on Signal Processing*, volume 65, no. 7, Apr. 2017: pp. 1795–1809.

[6] K. Dedecius; V. Sečkárová. Factorized estimation of partially shared parameters in diffusion networks. *IEEE Transactions on Signal Processing*, volume 65, no. 19, Oct. 2017: pp. 5153–5163.

[7] A. H. Sayed. Adaptation, Learning, and Optimization over Networks. *Foundations and Trends® in Machine Learning*, volume 7, no. 4-5, 2014: pp. 311–801.

[8] A. H. Sayed. Adaptive networks. *Proceedings of the IEEE*, volume 102, no. 4, Apr. 2014: pp. 460–497.

[9] Y. Yao; A. Vehtari; et al. Using stacking to average Bayesian predictive distributions. 2017.

[10] S. J. Taylor; B. Letham. Forecasting at scale. Sept. 2017.

[11] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, volume 19, no. 6, 1974: pp. 716–723.

[12] M. Szydlowski; A. Krawiec; et al. AIC, BIC, Bayesian evidence against the interacting dark energy model. 2008.

[13] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, volume 6, no. 2, Mar. 1978: pp. 461–464.

[14] A. E. Raftery; M. Kárný; et al. Online prediction under model uncertainty via dynamic model averaging: Application to a cold rolling mill. *Technometrics*, volume 52, no. 1, Feb. 2010: pp. 52–66.

[15] L. Onorante; A. E. Raftery. Dynamic model averaging in large model spaces using dynamic Occam's window. 2014.

[16] T. McCormick; A. E. Raftery; et al. Dynamic logistic regression and dynamic model averaging for binary classification. *Biometrics*, volume 68, 08 2011: pp. 23–30.

[17] J. Smith; K. F. Wallis. A simple explanation of the forecast combination puzzle. *Oxford Bulletin of Economics and Statistics*, volume 71, no. 3, June 2009: pp. 331–355.

[18] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[19] W. McKinney. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference*, volume 445, Austin, TX, 2010, pp. 51–56.

[20] F. Bovo. Robustats documentation. [online], 2020.

[21] F. Pedregosa; G. Varoquaux; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[22] T. E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, volume 9, no. 3, 2007: pp. 10–20.

[23] S. Seabold; J. Perktold. Statsmodels: Econometric and statistical modeling with Python. In *9th Python in Science Conference*, 2010.

[24] J. Ellson; E. Gansner; et al. Graphviz — open source graph drawing tools. In *Lecture Notes in Computer Science*, Springer-Verlag, 2001, pp. 483–484.

[25] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, volume 9, no. 3, 2007: pp. 90–95.

[26] F. Pérez; B. E. Granger. IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, volume 9, no. 3, May 2007: pp. 21–29, ISSN 1521-9615.

[27] P. H. Garthwaite; E. Mubwandarikwa. Selection of prior weights for weighted model averaging. *Australian & New Zealand Journal of Statistics*, volume 52, no. 4, Nov. 2010: pp. 363–382.

Appendix A

Glossary

The terms listed below are used in the thesis. Their more concrete definitions are in the particular parts of the text.

A priori In advance, supposed

Agent Independent component of the network, in most cases used interchangeably with *node*

AIC Akaike information criterion

BIC Bayesian information criterion

Closed neighbourhood Node and its neighbours

Collaborative modelling Modelling while exchanging information between multiple agents in order to create better predictions

Configuration Network and agent properties set before the experiment

Heterogeneous models Models using different inner methods and/or parameters (which are not to be shared) in order to predict the process

MAE Mean absolute error

Model Abstraction of the process aiming to describe its behaviour and possibly predict it

Module Standalone Python code, stored in a file with *.py* suffix

MSE Mean squared error

Neighbour Another node with which the information can be directly shared by the node

Node Independent component of the network, in most cases used interchangeably with *agent*

Node degree Number of neighbours of the node

Observation Value of the process available to the agent

Open neighbourhood Node's neighbours (node itself excluded)

Prediction Agent's estimate of the next value of the process

Weight Importance of (agent's or one of its neighbours') predictions considered by the agent

Appendix B

Contents of enclosed DVD

- *readme.txt* ..... the file with DVD contents description
- src ..... source codes dir
  - data ..... dir with a downloaded .csv data file
  - doc ..... dir with simple HTML documentation
  - jup ..... dir with all run notebooks
  - thesis ..... dir of LaTeX source codes of the thesis
  - *model_fusion.py* ..... implemented Python module
  - *Process_1.ipynb* ..... sample notebook, generated data
  - *Process_2.ipynb* ..... sample notebook, real data
- text ..... the thesis text dir
  - *MT_Smid_Martin_thesis.pdf* ..... the thesis text in PDF format