Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Maria Soleim

Reproducibility of the Top-Performing Methods in the M4 Competition

Master's thesis in Computer Science
Supervisor: Odd Erik Gundersen
June 2020

NTNU
Norwegian University of Science and Technology

Maria Soleim

Reproducibility of the Top-Performing Methods in the M4 Competition

Master's thesis in Computer Science
Supervisor: Odd Erik Gundersen
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Reproducibility has recently received increased attention within artificial intelligence. Although it is claimed that artificial intelligence is having a reproducibility crisis, this is not yet confirmed about time series forecasting. This study aims to determine to what degree today's research within the field of time series prediction is reproducible. An attempt to reproduce some of the methods from the M4 competition could fill this gap in the literature. Ten of the top-performing methods in the M4 competition have been attempted reproduced. The eight methods that were successfully rerun produced forecasts that were not equal to the original submissions, but still gave a score that did not change the order of the top-performing methods in the competition.

Sammendrag

Reproduserbarhet har nylig fått økt oppmerksomhet innen kunstig intelligens. Selv om det hevdes at kunstig intelligens har en reproduserbarhetskrise, er dette ennå ikke bekreftet om tidsserieprognoser. Denne studien tar sikte på å bestemme i hvilken grad dagens forskning innen tidsserieprognoser er reproduserbar. Et forsøk på å reprodusere noen av metodene fra M4-konkurransen kunne fylle dette gapet i litteraturen. Ti av de beste resultatene i M4-konkurransen er blitt forsøkt reprodusert. De åtte metodene som vellykket ble kjørt om igjen produserte prognoser som ikke var like de originale innleveringene, men ga en poengsum som ikke endret rekkefølgen på resultatlista i konkurransen.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

This research is spurred by the experience that a lot of recent artificial intelligence (AI) research has turned out to be difficult to reproduce. Reproducibility is a cornerstone of the scientific method. Through reproducibility, researchers can confirm and build upon each other's work. Not all research is perfect and some hypotheses must be discarded in an attempt to reproduce an experiment. When a research paper cannot be reproduced, there is no evidence that the results are correct and the paper is simply worth nothing. Reproducibility has gained increased attention in the last years in a variety of fields like psychology (Aarts et al., 2016), medicine (Stupple et al., 2019; Niven et al., 2018), and also in AI.

It has turned out that many AI experiments are not even possible to recreate due to poor documentation (Dacrema et al., 2019; Gundersen and Kjensmo, 2018). Some reports are missing critical information such as parameters, source code or data. Even if a model is well documented, there could be sources to non-determinism like different initialization of variables or data examples or the order of the data presented during training. Also information about hardware and compilation settings used might be crucial for the result due to the way floating-point operations are performed. Last, but not least, when comparing novel models against baselines, it is not uncommon that poor baselines are used (Dacrema et al., 2019). They might not be optimized or they might not be the current state-of-the-art at the specific task. As a result, the proposed method is claimed to be the new state-of-the-art. We can say that AI is having a *phantom progress*.

During this work, I will investigate if reproducibility is a problem in the field of time series forecasting. Similar contributions have been done in other machine learning fields like

GANs (Lucic et al., 2018), deep reinforcement learning (Lynnerup and Hallam, 2019), and image classification (Bouthillier et al., 2019). Melis et al. (2017) has conducted a relevant study on NLP, concluding that vanilla LSTM might be as good as recently published forecasting methods. To the best of my knowledge, no other studies have so far examined the reproducibility of forecasting architectures.

1.2 Problem Outline

In this project, I will attempt to reproduce some of the methods from the M4 competition, the most extensive time series competition to date. As the main goal of the M4 competition is to discover new forecasting methods, it is a competition with a strong focus on reproducibility. Even a reproducibility prize is awarded to one of the methods. If there are somewhere in the field of time series forecasting where the methods should be reproducible, it is in this competition. There were 49 valid submissions, which were compared and ranked from best to worst. Many of the models are publicly available, as well as the training and test data and the results of every model.

1.3 Research Context

This work is my master's thesis for my degree in Computer Science at the Norwegian University of Science and Technology (NTNU). The project builds upon work from the preliminary specialization project written by the author of this thesis (Soleim, 2019). The project was carried out at TrønderEnergi under the supervision of Odd Erik Gundersen, associate professor at the Department of Computer Science at NTNU and chief AI officer at TrønderEnergi.

1.4 Objective and Research Questions

The objective of this master's thesis is to ensure that future research within the field of time series forecasting is reproducible. Three research questions are formulated:

1. To what degree is the top-performing methods in the M4 competition reproducible?
2. Which factors make research on time series forecasting difficult to reproduce?
3. How can we work for future research on time series forecasting to reach a higher level of reproducibility?

Chapter 2

Background

2.1 Time Series

A time series is a series of observations taken sequentially over time (Pole et al., 1994). Typically, successive observations are done with equal spacing in between each observation. The time between each interval is called the *resolution* of the series. For example, we could measure the height of the ocean tides every hour, or we could measure the air pollution once every day.

2.1.1 Forecasting

When *forecasting*, we are trying to predict the future values of a time series as precisely as possible, given all of the information available (Hyndman and Athanasopoulos, 2018). Given a time series of the number of people visiting the gym each day for two weeks, we could try to predict the number of people visiting the gym each day for the upcoming week. The *forecasting horizon* is the number of timesteps ahead that we try to predict. In this case, we will have a horizon of seven. In *point forecasting*, we predict the most likely value for each timestep in the forecasting horizon. For example, we could predict that there will be 112 visitors at the gym on Monday, 87 on Tuesday, and so on. Another kind of forecasting is by using a *prediction interval*. That is, instead of predicting one value, we could predict an interval in which the real value is likely to be contained. In the gym example, a 95% prediction interval could be that on Monday there will be between 100 and 124 visitors, Tuesday there will be between 77 and 97 visitors etc. Figure 2.1 shows the difference between a point forecast and a prediction interval.

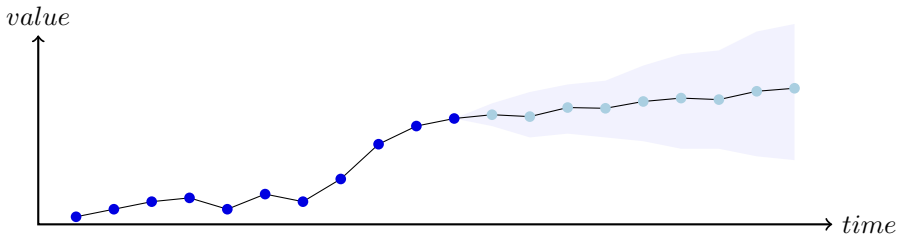


Figure 2.1: A time series can be predicted with point forecast or prediction interval. The light blue dots shows a point forecast. A specific value is predicted as the most likely value. The light blue shade behind the light blue dots shows a prediction interval. The real value is most likely inside this area.

2.1.2 Models

A model that predicts future values of a time series using only the previous values of that series is a pure *time series model*. However, it is very often not only the previous values in a time series that affect its future. Usually, there are *predictor variables* influencing a future value. Say we have a time series of how much popcorn a cinema has sold every day, and we want to predict how much popcorn the cinema will sell tomorrow. People may go more to the movies and buy more popcorn in the weekend or holidays. Cold temperature outside or generally bad weather may attract people to the cinema and consequently boost the popcorn sale. A new movie coming out could appeal to the community. Some of these factors could be time series themselves. For instance, one series could tell if it is a weekend by giving the value one on Saturdays and Sundays and zero otherwise. Another time series could tell if it is a holiday, and a third time series could tell the temperature outside. Tomorrow's value of those predictor variables could be predicted to then be the input to a model that foresees tomorrow's popcorn sales. Other variables are information that the cinema employees know, like if there is a new movie coming out. This information could also be input to the same model. The popcorn sales from the past could be included in the model or not. If we leave it out, we call it an *explanatory model*. Otherwise, we have a *mixed model*.

2.1.3 Patterns

A time series could follow several patterns. A *trend* exists when there is a long-term increase or decrease in the data (Hyndman and Athanasopoulos, 2018). When a trend pattern exhibits a general direction that is upwards, where there are higher highs and higher lows, like in figure 2.2, there is an increasing trend. In the same way, when there is a gradual movement to relatively lower highs and lower lows, there is a decreasing trend. A trend is not necessarily linear - it could also be exponential, like in the figure. Even if a series segment has no increasing or decreasing trend, we could call that a horizontal or a stationary trend.

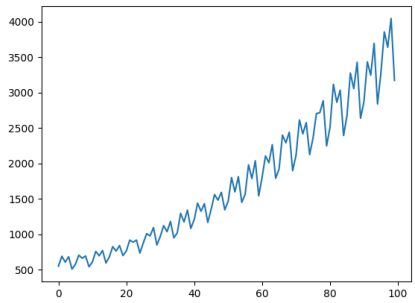


Figure 2.2: A time series generated to illustrate an increasing trend.

A *seasonal pattern* occurs when a series is affected by seasonal variations such as time of the year or day of the week. The *frequency* is the number of observations before the pattern repeats. These observations are called a *year*. Each timestep in a year is of a different *season*. The seasons repeat every year. This is easy to understand if you have a time series where you measure something four times a year. Then the calendar year is a year and the seasons are summer, fall, winter, and spring. But a year does not have to be a calendar year and the seasons could be different. For example, a zoo could measure the number of visitors each day. They could experience a high number of visitors on the weekend and a lower number of visitors the other days of the week. Consequently, there is a seasonal pattern of seven days. In other words, one regular week is called a year. Each day of the week is a season. A time series is not restricted to only have one seasonal pattern. For instance, we can measure the temperature once every hour. Since the temperature at noon today will be about the same temperature as noon yesterday, there will be a frequency of 24 timesteps. Yet, the temperature at noon today will also be about the same as the temperature at noon one year ago. 24 timesteps each day and a year having a mean of 365.24 days yields a frequency of $24 \times 365.24 \approx 8766$. As a result, there are not only one, but two seasonal patterns in this time series.

A *cyclic pattern* occurs when the data points increases and decreases without any fixed frequency. It must not be confused with seasonal patterns. In a time series with a seasonal pattern, we can divide the series into consecutive subintervals of equal length (the length is the frequency), which will have peaks at about the same data point in every subinterval. For example, the temperature throughout the year will be the highest sometime in the summer half of the year. However, the highest measured temperature will never occur in January in the northern hemisphere, which could have been the case if the temperature had a cyclical pattern. To clarify, seasonal patterns have a fixed length of the repeating pattern, whereas cyclical patterns can vary widely in duration. The stock market typically has cyclical patterns.

As well as systematic components like trends, seasonal patterns, and cyclic patterns, a time series contains noise, which simply is random variation in the series. More specifically, we

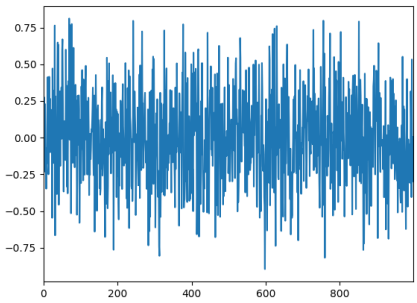


Figure 2.3: A time series generated to simulate white noise.

have *white noise*. A time series *is* white noise if it has a mean equal to zero, the standard deviation is constant over time and the correlation between variables is zero. Such a series is shown in figure 2.3. White noise can by definition not be predicted. When you are doing time series forecasting, the difference between the predicted values and the real values is the error of the forecast. This error should ideally be close to white noise. When the error is white noise, it is impossible to do any better prediction, because the values are random.

2.1.4 Simple Forecasting Methods

The Naïve Method

To predict a value in a series, we can simply use the last observed value in the series as our prediction. For instance, a zoo could keep track of the number of visitors every day. To predict how many visitors the zoo will have tomorrow, they could simply look at the number of visitors from the last day and use that number as the prediction. The day after tomorrow will have the same prediction and so will every day in the future. The equation for a forecast given by the Naïve method is

$$\hat{y}_t = y_{t-n}, \tag{2.1}$$

where y_t is the observed value at timestep t and n number of timesteps since the last observed value. Notice that t is not a variable in the forecast. The forecast is the same for all steps in the horizon. This is what we call a *flat forecast*.

The Seasonal Naïve Method

If a time series is having a seasonal pattern, one could use the last observed value of the same season as the new prediction. For instance, if tomorrow is a Saturday, the number of people in the zoo last Saturday could be used as a prediction. The prediction of Sunday will

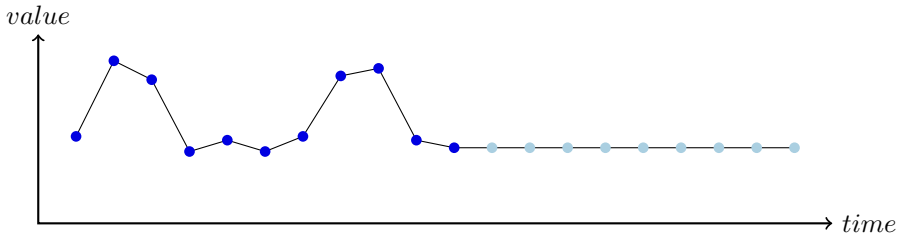


Figure 2.4: Forecasting with the naïve method. The dark blue dots indicate observed values and the light blue dots indicate the predicted values. Every prediction equals the last observed value.

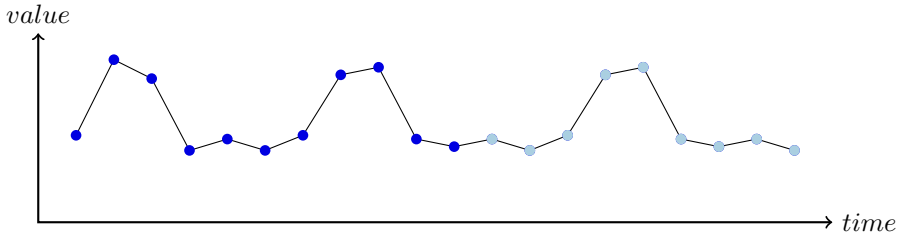


Figure 2.5: Forecasting a time series with frequency 7 using the seasonal naïve method. The dark blue dots indicate observed values and the light blue dots indicate the predicted values. Every prediction equals the last observed value of the same season.

have the same prediction as last Sunday. The next Sunday will have the same prediction. The prediction of a value $\hat{y}_t$ is given by the formula

$$\hat{y}_t = y_{t-k*m},$$

where m is the frequency and k is the number of commenced years since the last observed value. In the zoo example, the year is one week and the frequency is seven. When there is less than one week since the last observed value, $k = 1$ and $\hat{y}_t = y_{t-m} = y_{t-7}$, so that the prediction is the value from seven days ago. If there is between one and two weeks since the last observed value $k = 2$ and $\hat{y}_t = y_{t-2*m} = y_{t-2*7} = y_{t-14}$, so that the prediction is the value from 14 days before.

Exponential Smoothing

Exponential smoothing (ES) algorithms use a weighted average of past observations to predict the future. The weights are larger for recent observations and shrinks exponentially for earlier observations.

One form of ES is *simple exponential smoothing* (SES). It is useful when the time series

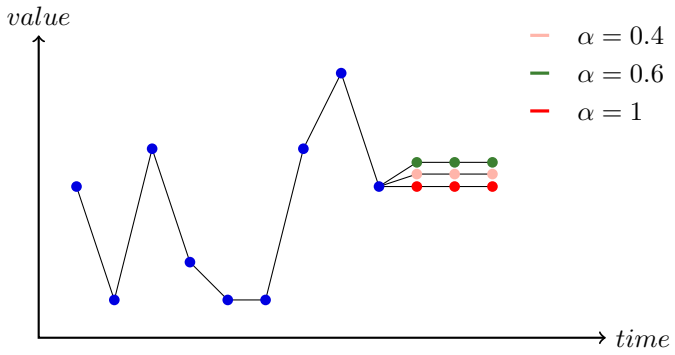


Figure 2.6: Forecasting a time series with SES. The blue dots indicate observed values, while the green, yellow and red dots indicate different forecasts. Different values of α gives different weights to the earlier observed values and make different forecasts. The forecast does not necessarily increase or decrease with an increasing α. $\alpha = 1$ give a forecast equal to the last observed value. $\alpha = 0.6$ gives more weight to earlier values and in this case, it gives a higher forecast than $\alpha = 1$. $\alpha = 0.4$ gives even more weight to the early observations and in this case that gives a forecast lower than $\alpha = 0.6$ and higher than $\alpha = 1$. SES gives a *flat* forecast: the forecast is the same for every timestep in the future.

has no clear trend or season. A forecast using SES is done with the following equation:

$$\begin{aligned}\hat{y}_t &= \alpha * y_n + \alpha * (1-\alpha) * y_{n-1} + \alpha * (1-\alpha)^2 * y_{n-2} ... \\ &= \sum_{x=1}^{n} \alpha * (1-\alpha)^{x-1} * y_{n+1-x},\end{aligned}$$

where n is the number of observed timesteps in the series and α is a *smoothing parameter* in the range from zero to one. If $\alpha = 1$, then all terms become 0 except for the first and the forecast is equal to the forecast of the seasonal naïve method. As α approaches zero, more weight is added to observations from the past. Notice that t is not a variable in the forecast. Thus, SES produces a flat forecast.

Another special kind of ES is Holt-Winters' multiplicative method. This method is able to capture seasonality. There is one forecast equation and three smoothing equations for the level ℓ, trend b_t, and seasonal component s_t:

$$\hat{y}_{t+h|h} = (\ell_t + hb_t)s_{t+h-m(k+1)} \tag{2.2}$$

$$\ell_t = \alpha * \frac{y_t}{s_{t-m}} + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{2.3}$$

$$b_t = \beta * (\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \tag{2.4}$$

$$s_t = \gamma * \frac{y_t}{\ell_{t-1} + b_{t-1}} + (1-\gamma)s_{t-m} \tag{2.5}$$

The equation 2.2 is the forecast equation and the equations 2.3-2.5 are the smoothing equations. α, β, and γ are smoothing parameters. The level ℓ_t, is an estimate of the level of the series at time t. The trend b_t denotes an estimate of the change in the series at time t. The seasonality s_t denotes seasonal variations and is expressed in relative terms to the series. Within each year, the seasonal component will sum up to approximately m. This means that we can divide all values in the series with the seasonal component to *deseasonalize* the series.

2.1.5 Time Series Decomposition

Hyndman and Athanasopoulos (2018) describes how a time series Y can be divided into different components: a trend component T, a seasonal component S and a remainder component R. These components are series themselves. Decomposition could be done either additive or multiplicative. With additive decomposition, each timestep in Y, y_t, equals the sum of the three components Y_t, S_t and R_t. With multiplicative decomposition, each timestep equals the product of the three components. This tool can be used for analyzing a time series as well as a preparation step before forecasting.

$$\textbf{Additive decomposition} \qquad y_t = T_t + S_t + R_t \tag{2.6}$$

$$\textbf{Multiplicative decomposition} \qquad y_t = T_t * S_t * R_t \tag{2.7}$$

Detrending

To detrend means to remove the trend component from a series. We do this by finding the trend component T and then removing it from the time series. If we use additive decomposition, we remove it by subtracting it from the series and if we use a multiplicative decomposition we divide the series with the trend component.

The method for obtaining the trend component is the same for additive and multiplicative decomposition. For every step y_t in the time series, the average is taken over a year of data. One value from each season is included in the average. An equal number of timesteps before the given timestep and after the current timestep is used. For instance, a time series could have daily observations and a weekly pattern. Step four is a Thursday. The trend value of step four T_4 is the average of the whole week. The trend value for step five T_5 will be the average of the steps 2 to 8, namely Tuesday to Monday. We do this with each step of the series, except for the very earliest steps and the very latest steps, which do not have enough steps on each side to average. This way a new time series is obtained and this is what we call the trend.

If the frequency is an even number, however, it is not possible to pick an equal number of steps on each side of y_t. For instance, a series could have quarterly observations and a yearly pattern. For the summer step, should we average winter to autumn or spring to

winter? For the season that is furthest away from y_t in time, in this case, winter, we could either pick the value preceding y_t, or we could pick the value following y_t. They are both the same distance away from y_t. That is y_{t-k} and y_{t+k} where $k = f/2$. We solve this by picking half of the preceding one and half of the following one.

Formally, we can say that each value in the trend is computed the following way:

$$T_t = \begin{cases} \frac{1}{f} \sum_{j=-k}^{k} y_{t+j}, & \text{when } f \text{ is odd} \\ \frac{1}{f} \left(\frac{1}{2} y_{t-k} + \frac{1}{2} y_{t+k} + \sum_{j=-k+1}^{k-1} y_{t+j} \right), & \text{when } f \text{ is even} \end{cases}$$

where f is the frequency of the series and $k = \lfloor \frac{f}{2} \rfloor$.

Figure 2.7a shows a series along with its trend component and how it becomes when it is detrended with additive decomposition. Note how the detrended series keeps close to the x-axis. It does not tend to move upwards or downwards like a trended series. The trend component, however, keeps close to the original series, but is smoother.

Deseasonalizing

Time series having a seasonal pattern can be deseasonalized. By deseasonalizing a time series, we remove the seasons, such that the mean within each season is stationary. For instance, the mean of all spring values will equal the mean of all summer values in a time series with a quarterly pattern. First, the seasonal component is calculated. That is done by first finding the average value of every season. Then create a time series equal to the original series, but substitute all values with the average value for that season. The newly created series is shifted to obtain the seasonal component. In additive decomposition, the series is shifted to add up to 0, whereas in multiplicative decomposition, it is shifted to add up to the frequency. To obtain the deseasonalized series we either subtract or divide with the seasonal component depending on whether we are doing additive or multiplicative decomposition. An example of the procedure follows.

Imagine we have the time series

$$x = [\ 6 \quad 13 \quad 8 \quad 6 \quad 8 \quad 14 \quad 8 \quad 5 \quad 7 \quad 15\ ]$$

with a quarterly pattern, hence the frequency is 4. The average of each season. Given that

the first value in the series is spring, the next summer and so on, we have

$$\mu_{\text{spring}} = \frac{6+8+7}{3} = 7$$
$$\mu_{\text{summer}} = \frac{13+14+15}{3} = 14$$
$$\mu_{\text{fall}} = \frac{8+8}{2} = 8$$
$$\mu_{\text{winter}} = \frac{6+5}{2} = 6.5$$

We create a new time series with the average of all seasons:

$$averages = [\ 7 \quad 14 \quad 8 \quad 5.5 \quad 7 \quad 14 \quad 8 \quad 5.5 \quad 7 \quad 14\ ]$$

Now we want to shift the series. For additive decomposition, we shift the averages such that they add up to zero. We do this by subtracting the average of the series. The average of this series is 9. Subtracting 9 from all averages gives us

$$season = [\ -2 \quad 5 \quad -1 \quad -3.5 \quad -2 \quad 5 \quad -1 \quad -3.5 \quad -2 \quad 5\ ].$$

To obtain the deseasonalized series we subtract the seasonal component from the original series.

$$deseasonalized = [\ 8 \quad 8 \quad 9 \quad 9.5 \quad 10 \quad 9 \quad 9 \quad 8.5 \quad 9 \quad 10\ ]$$

Notice that the average value for every season is nine for every season. For multiplicative series, we shift the averages such that they add up to the frequency of the series. We do that by subtracting the number we get when we add up all the numbers in the series and subtract the frequency and then divide it with the length of the series.

$$\text{subtrahend} = \frac{\text{sum(series)} - \text{frequency}}{\text{length(series)}} = \frac{90-4}{10} = 8.6$$

Dividing the averages with 8.6, the season becomes

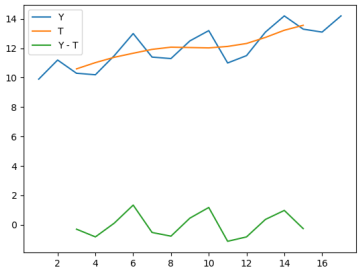
$$season = [\ -1.6 \quad 5.4 \quad -0.6 \quad -3.1 \quad -1.6 \quad 5.4 \quad -0.6 \quad -3.1 \quad -1.6 \quad 5.4\ ].$$

We divide with the seasonal component to obtain the deseasonalized series:

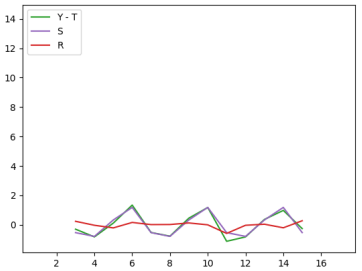
$$deseasonalized = [\ 7.6 \quad 7.6 \quad 8.6 \quad 9.1 \quad 9.6 \quad 8.6 \quad 8.6 \quad 8.1 \quad 8.6 \quad 9.6\ ]$$

Also here, the average value for every season is the same, this time 8.6.

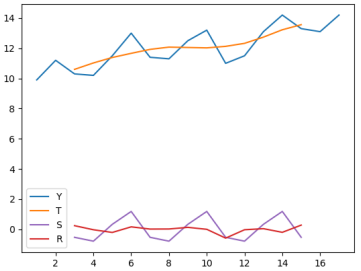
If we deseasonalize an already detrended series, we are left with the remainder component. Figure 2.7b shows the earlier detrended series with its seasonal component and the remainder component. Figure 2.7c shows the trend component, the seasonal component, and the remainder component, which all add up to the original time series.



(a) The original time series Y is detrended with additive decomposition. The trend component T is first calculated. Then the trend is subtracted from the original series to obtain a detrended series $Y - T$.



(b) The previously detrended series is deseasonalized.



(c) The components T, S and R add up to the original series Y.

Figure 2.7: A time series Y is decomposed into a trend component T, a seasonal component S and a remainder component R. In this example, additive decomposition is used. That means that T, S and R adds up to Y.

2.1.6 The Random Walk Model

The random walk model is a simple, but important model in time series forecasting. The model assumes that every value in a time series takes a random step away from the previous value. The steps are independently and identically distributed in size. A rational forecast for a random walk would be a naïve forecast, like in equation 2.1. The series is equally likely to go up or down. Therefore a naïve forecast is also called the random walk forecast.

A random walk could also have a *drift*. That happens if the average of the random steps away from the previous value is non-zero. If this average is positive, the time series will have a drift upwards. For instance, if the average value of the random step is 0.2, then the time series will on average increase with 0.2 for each timestep. If the average step away from the previous is negative, then the time series will have a drift downwards. In both cases, the naïve forecast will not be best. If the average increase from one value to another is d, then a rational forecast for step $t + h$ would be

$$\hat{Y}_{t+h} = Y_t + h * d.$$

Figure 2.8 shows two random walks, one with drift and one without, and their forecasts. Both are going up and down, but in the long term, the series without drift is about the same value that it was at $t = 0$. The random walk with drift has increased from 0 to 200 in 1000 steps which indicates an average increase in $d = 0.2$ each step.

2.1.7 ARIMA

The ARIMA method is a combination of three other methods: differencing, autoregression, and moving average model. A description of each of them follows.

Differencing

The method is based on comparing each observation with the value of the observation of the same season, one year earlier. The value of the observation a year earlier is subtracted from the value of the observation this year.

For instance, we could have a time series showing how many waffles that are sold from a waffle stand every day, like shown in figure 2.9. The waffle sale has a weekly seasonality, each day of the week being a season. To deseasonalize the series, we subtract the number of waffles sold every Monday with the number of waffles sold one Monday earlier. For instance, the value at Monday 8th is 6. The value of the same season, one year earlier, namely Monday 1st, has the value 4. The value of Monday 8th is then $6 - 4 = 2$. The same is done with the other days of the week. Figure 2.10 shows the differenced series. This will create a new time series, which is more likely to be seasonal stationary. If the new time series is not seasonal stationary, it could be differenced itself. Then we have second-order differencing. In practice, you will probably never have use for a higher order

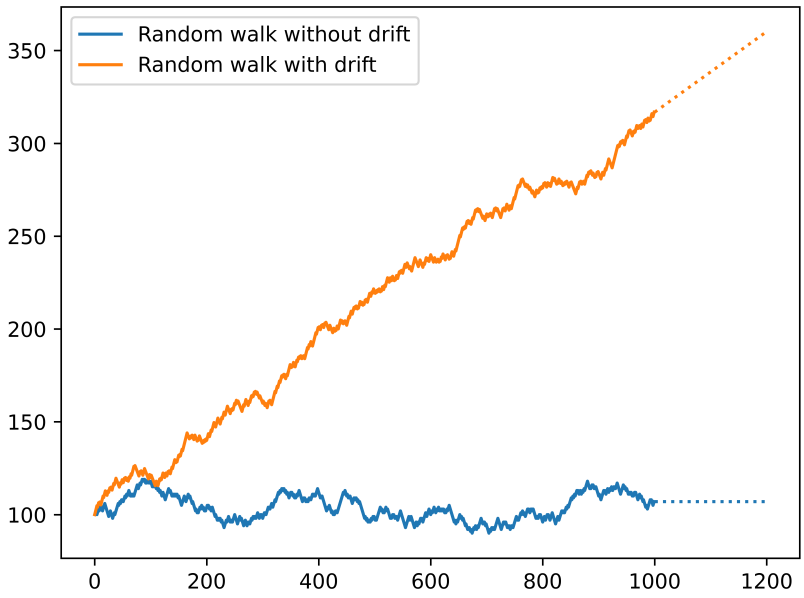


Figure 2.8: A series that acts like a random walk without drift and a series that acts like a random walk with drift.

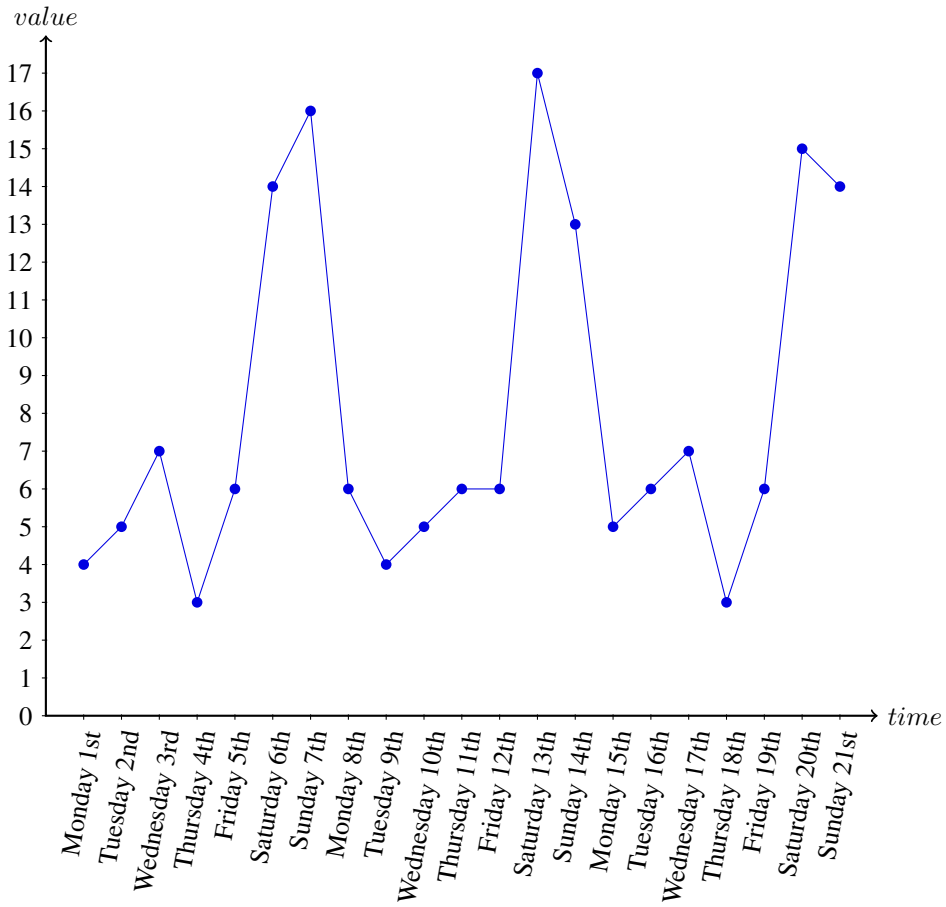


Figure 2.9: A time series showing the number of waffles sold each day. The series seems to have a weekly pattern.

than second-order differencing. For each round of differencing, the new time series will not have any data for the first year.

Autoregressive Models

Autoregression is simply forecasting using a linear combination of past values of the variable. An autoregressive model of order p can be written as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t$$

where ϵ_t is white noise.

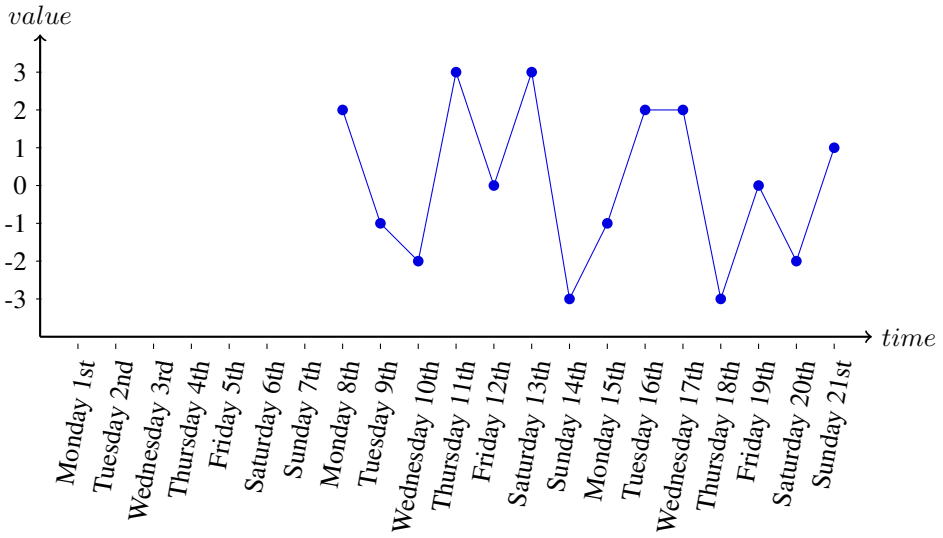


Figure 2.10: The waffle sale time series from figure 2.9 is differenced.

Moving Average Models

A moving average model of order q can be written as

$$y_t = c + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + ... + \theta_q\epsilon_{t-q}.$$

Combining the Methods to ARIMA

The three methods are combined to the following formula:

$$y'_t = c + \phi_1 y'_{t-1} + ... + \phi_p y'_{t-p} + \theta_1\epsilon_{t-1} + ... + \theta_q\epsilon_{t-q} + \epsilon_t,$$

where y'_t is the differenced series. It could be differenced zero, once or several times. We can write ARIMA(p, d, q) where

- p is the order of the autoregressive part
- d is the order of differencing
- q is the order of the moving average

2.1.8 ETS

Hyndman et al. (2008) describes the concept of ETS. ETS is a collective name for different kinds of ES algorithms. The letters E, T, and S stands for Error, Trend, and Seasonal.

They can take different values. Error could have the values ”Additive” or ”Multiplicative”. Trend could take the values ”None”, ”Additive” or ”Additive damped”. Seasonal could take the values ”None”, ”Additive” or ”Multiplicative”. This means that there is $2 * 3 * 3 = 18$ different kinds of ETS methods. We usually write them ETS(X, X, X) with the first X indicating which kind of error there is, the second describing the trend and the last describing the seasonal. For instance, we could have ETS(A, N, N) which indicates additive error, no trend, and no seasonal. That is in fact SES with additive error as described in section 2.1.4. By decomposing a time series like in section 2.1.5, we can use different kinds of ETS-methods.

2.2 The M4 Competition

The M4 competition was held in 2018 and was the fourth competition in the series of M competitions (M4Team, 2019). The main goal of these competitions is to discover methods to predict the future. The purpose of the M4 competition was to replicate the results of the previous three ones and extend them into two directions. Firstly, increasing the number of series to 100,000. Secondly, including machine learning methods. The competition is divided in two. One part of the competition is about making the best point prediction. The other is about making the best 95% prediction interval. This research will only focus on the point prediction part of the competition.

2.2.1 The M4 Competition Dataset

The dataset consists of 100000 time series of different resolutions and origins as seen in table 2.1. The length of the series varies widely. For instance, the yearly series has lengths between 13 and 835. The horizon for the contestants to predict depends on the resolution of the time series and could be seen in table 2.2. For example, a yearly time series would require a forecast of 6 points ahead in time. The dataset is given in six CSV-files, one for each resolution type. Each line represents a time series. The first comma separated value in the line gives an id of the time series, and the following values are points in the time series. In addition, there is an info file that for each series tells the origin of the series (demographic, finance, etc.), as well as the resolution (yearly, quarterly, etc.).

2.2.2 Performance Measures

The performance measure chosen to evaluate the point predictions was a combination of two popular accuracy measures: symmetric mean absolute percentage error (sMAPE) (Makridakis, 1993) and mean absolute scaled error (MASE) (Hyndman and Koehler, 2006).

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	1088	6519	3716	3903	6538	1236	**23000**
Quarterly	1858	5305	4637	5315	6020	865	**24000**
Monthly	5728	10987	10017	10016	10975	277	**48000**
Weekly	24	164	6	41	112	12	**359**
Daily	10	1559	422	127	1476	633	**4227**
Hourly	0	0	0	0	0	414	**414**
Total	**8708**	**24534**	**18798**	**19402**	**25121**	**3437**	**100000**

Table 2.1: The number of time series in the M4 competition based on their resolution and origin.

Resolution	Horizon
Yearly	6
Quarterly	8
Monthly	18
Weekly	13
Daily	14
Hourly	48

Table 2.2: The horizon to predict in the M4 competition given the resolution of the time series.

sMAPE

To understand the performance measure sMAPE, it might be convenient to first have a look at the simpler function *symmetric absolute percentage error* (sAPE):

$$\text{sAPE} = \frac{|Y_t - \hat{Y}_t|}{\frac{1}{2}(|\hat{Y}_t| + |\hat{Y}_t|)} * 100\%,$$

where Y_t is the real value at point t and $\hat{Y}_t$ is the predicted value at point t. In the numerator, $|Y_t - \hat{Y}_t|$ is the absolute error between a real value and a predicted value. The absolute error could be used as a performance measure by itself, but it would provide larger error for time series with high values, and opposite with time series with values close to zero. That is unfavorable when there are time series from various domains. Therefore, the absolute error is divided with the average of the predicted value and the real value $\frac{1}{2}(|Y_t| + |\hat{Y}_t)|)$. Multiplying this with 100%, we get a measure of how large the error is relative to how large the numbers actually are. For example, if the real value $Y_t = 100$ and we make the prediction $\hat{Y}_t = 150$, the error for this individual point will be

$$\text{sAPE} = \frac{|100 - 150|}{\frac{1}{2}(|100| + |150|)} * 100\% = \frac{50}{125} * 100\% = 40\%.$$

If the prediction equals the actual value, then $|Y_t - \hat{Y}_t| = 0$, resulting in sAPE = 0, which is the best sAPE value. The larger sAPE, the worse is the prediction.

The sAPE metric is useful for calculating the difference between one forecasted value and the real value. In the M4 competition, the participants are supposed to predict not only one value for each time series, put several values given by the forecasting horizon. To put a number on the quality of a model that predicts several steps into the future, we can average the sAPE values. That gives us the sMAPE:

$$\text{sMAPE} = \frac{1}{h}\sum_{t=n+1}^{n+h}\frac{|Y_t - \hat{Y}_t|}{\frac{1}{2}(|Y_t| + |\hat{Y}_t|)} * 100\%,$$

where n is the number of timesteps in the training set and h is the number of steps in the forecasting horizon. Notice that the summation starts on timestep $n+1$ in the series, which is the first timestep in the forecasting horizon, and ends at $n+h$, which is the last timestep in the horizon. Hence, the error is summed up over all predicted values and divided with the number of points in the forecasting horizon. In other words, the average relative error is found. The numerator and the denominator could be multiplied with 2, and the 2 could be pulled out of the summation to obtain the formula as it is usually written:

$$\text{sMAPE} = \frac{2}{h}\sum_{t=n+1}^{n+h}\frac{|Y_t - \hat{Y}_t|}{|Y_t| + |\hat{Y}_t|} * 100\%. \tag{2.8}$$

MASE

In order to calculate sMAPE, all we needed was the predicted forecast and the real values for the forecast. MASE, on the other hand, takes into account the values from the training set as well as the frequency for the series. It can be discussed how to find the frequency of a series. For the competition, the organizers decided which frequency that should be used for each type of resolution. These are shown in table 2.3. The participants were encouraged to explore other frequencies in the series, but these exact frequencies were used for the evaluation. First, we take a look at the simpler function *absolute scaled error* (ASE):

$$\text{ASE} = \frac{|Y_t - \hat{Y}_t|}{\frac{1}{n-m}\sum_{t=m+1}^{n}|Y_t - Y_{t-m}|},$$

where m is the frequency considered by the organizer as in table 2.3. In the numerator, we have the absolute error between the predicted and the real value. In the denominator, there is a summation. Inside the summation, the difference between an actual value and the actual value m timesteps earlier is found. In other words, this is the difference between an actual value and the actual value of the last observed value with the same season. That is the absolute error if we used a seasonal naïve method and it is referred to as the *seasonal difference*. The seasonal difference could for example be the difference between the temperature the 1st of December this year and the 1st of December last year. The difference is averaged over all training values that have an earlier observation of the same season. Hence, the ASE is the absolute error in the forecast divided by the average seasonal difference in the training set. Put differently, ASE is a measure for how good we are doing

Resolution	Frequency
Yearly	1
Quarterly	4
Monthly	12
Weekly	1
Daily	1
Hourly	24

Table 2.3: The frequency for each resolution considered by the organizers. The frequency is used for calculating MASE, which again is used to calculate OWA.

compared to how good we could have been doing using seasonal naïve method. As with sAPE, if the prediction is correct then the ASE is zero, and the higher the ASE the worse an algorithm is doing. If the ASE is one we are doing just as good as a seasonal naïve method. If it is larger than one we are doing worse.

If we have several predicted values in the forecasting horizon, we use the average of all the absolute errors in the horizon and divide by the same denominator:

$$\text{MASE} = \frac{\frac{1}{h}\sum_{t=n+1}^{n+h}|Y_t - \hat{Y}_t|}{\frac{1}{n-m}\sum_{t=m+1}^{n}|Y_t - Y_{t-m}|}$$

That is the MASE. Pulling $\frac{1}{h}$ out of the fraction, we obtain the formula as it is more commonly known:

$$\text{MASE} = \frac{1}{h}\frac{\sum_{t=n+1}^{n+h}|Y_t - \hat{Y}_t|}{\frac{1}{n-m}\sum_{t=m+1}^{n}|Y_t - Y_{t-m}|} \tag{2.9}$$

This version of MASE differs from the version proposed by Hyndman and Koehler (2006), where m was set equal to 1 regardless of the frequency.

The combination of sMAPE and MASE is referred to as the overall weighted average (OWA). The sMAPE and MASE is calculated for every single time series before they are averaged over all series. Then the average sMAPE is divided by the average sMAPE of the baseline model Naïve2, programmed by the organizers of the competition, to achieve a relative sMAPE. The average MASE is divided by the average MASE of Naïve2 to achieve a relative MASE. OWA is then found by calculating the average of the relative sMAPE and the relative MASE. A model with OWA less than one is doing better than Naïve2 and a model with OWA larger than one is doing worse.

$$\text{OWA} = \frac{\text{relative sMAPE} + \text{relative MASE}}{2} \tag{2.10}$$

2.3 Relevant Methods from the M4 Competition

A description of the methods from the M4 competition that will be relevant later in the thesis is described here.

2.3.1 ES-RNN

The winner of the M4 competition was Slawek Smyl. His method *Exponential Smoothing Recurrent Neural Network* (ES-RNN) achieved an OWA of 0.821, hence it was 17.9% better than Naïve2. Like the name of the model says, it is a combination of ES and Recurrent Neural Network. It is not an ensemble, but what Smyl calls a *truly hybrid* model (Makridakis et al., 2019). All ES parameters, like the seasonality and smoothing coefficients, are fitted concurrently with the neural network weights. The method is global in the way that the network parameters are learned based on multiple series. At the same time, the model learns parameters locally, considering that each series has its own smoothing parameters, which are updated only based on the loss for that series. The method has received positive attention due to the novel combination of global and local learning (Barker, 2019). The algorithm is written in `C++`.

Since many time series are several hundreds or even thousands of timesteps long, the long series are chopped such that only the last part of the series are used. For example, the yearly time series, where some series are as long as 835, are cut down to a maximum length of 60. Daily series, which could be over 9000 timesteps, are cut down to be no longer than 13 weeks, equivalent to 91 timesteps. Each series is assigned twice to one out of five networks. That means that it is first assigned to one network, and then it is assigned to another network that may or may not be the same as the first. Hence, each network will be trained on about 40% of the series.

However, when a series has 91 timesteps, not all timesteps are sent through a network at once. An input window and an output window of fixed size are sliding over the series as shown in figure 2.11. The values in the input window are sent together into a network and are supposed to predict the values in the output window. Smyl has experimented with sizes of the input window and set one size for each resolution. That is, for instance, four for yearly and seven for daily series. The prediction horizon determines the size of the output window. These are the values succeeding the values in the input window. The first time a series is introduced to a network, the input window starts at the beginning of the (possibly cut) series. The values from the input window are sent through the network. The output from the network is compared to the values in the output window. The loss is calculated and noted. Then the window slides one timestep ahead in time and the process is repeated. First when the output window reaches the end of the series and the windows cannot slide any further, the network weights and the smoothing parameters (α and γ) are updated, using the average loss.

The values in the input window are preprocessed as shown in figure 2.12: each small part

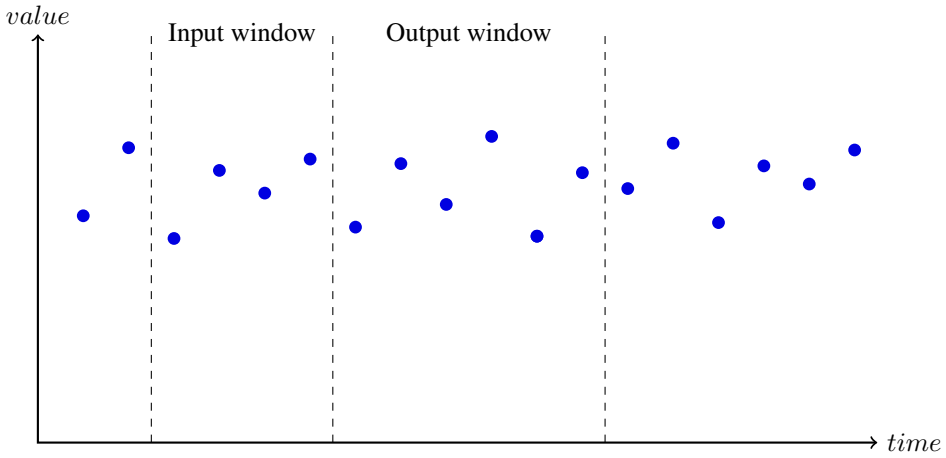


Figure 2.11: A yearly time series with input window size 4 and output window size 6, which are sliding over the series. The values in the input window are supposed to predict the values in the output window.

is deseasonalized, like described in section 2.1.5. Each kind of series, depending on its resolution, is treated with zero, one or two seasonalities. The figure shows a yearly series, which is treated with no seasonality and is therefore unaffected by this step. The deseasonalized values are *normalized*, by dividing with the level of the series. The logarithm is taken of every value and lastly, some noise is added. The preprocessed series is concatenated with a one-hot encoding of the origin of the series. The concatenated vector is sent through the network. The number of output nodes equals the prediction horizon required in the challenge.

The level of the series and the seasonal component are calculated the following way:

$$l_t = \alpha \frac{y_t}{s_t} + (1 - \alpha) * l_{t-1}$$

$$s_{t+m} = \gamma \frac{y_t}{l_t} + (1 - \gamma) * s_t,$$

which we recognize from Holt-Winters' multiplicative method. α and γ are smoothing coefficients between 0 and 1.

The values of the time series succeeding the input values, are the labels we are trying to predict. They are preprocessed in the same way as the input values, but without adding any noise. The output of the network is compared to the preprocessed labels using pinball loss. The pinball loss for one forecasted value is

$$pinBallLoss(y, \hat{y}) = \begin{cases} 2 * (y - \hat{y}) * \tau, & \text{if } y \geq \hat{y} \\ 2 * (y - \hat{y}) * (\tau - 1), & \text{otherwise,} \end{cases}$$

where y is the target value and $\hat{y}$ is the predicted value. If $\tau = 0.5$ the formula would have become $|y - \hat{y}|$ for all values of y and $\hat{y}$. Setting the value of τ to another value than

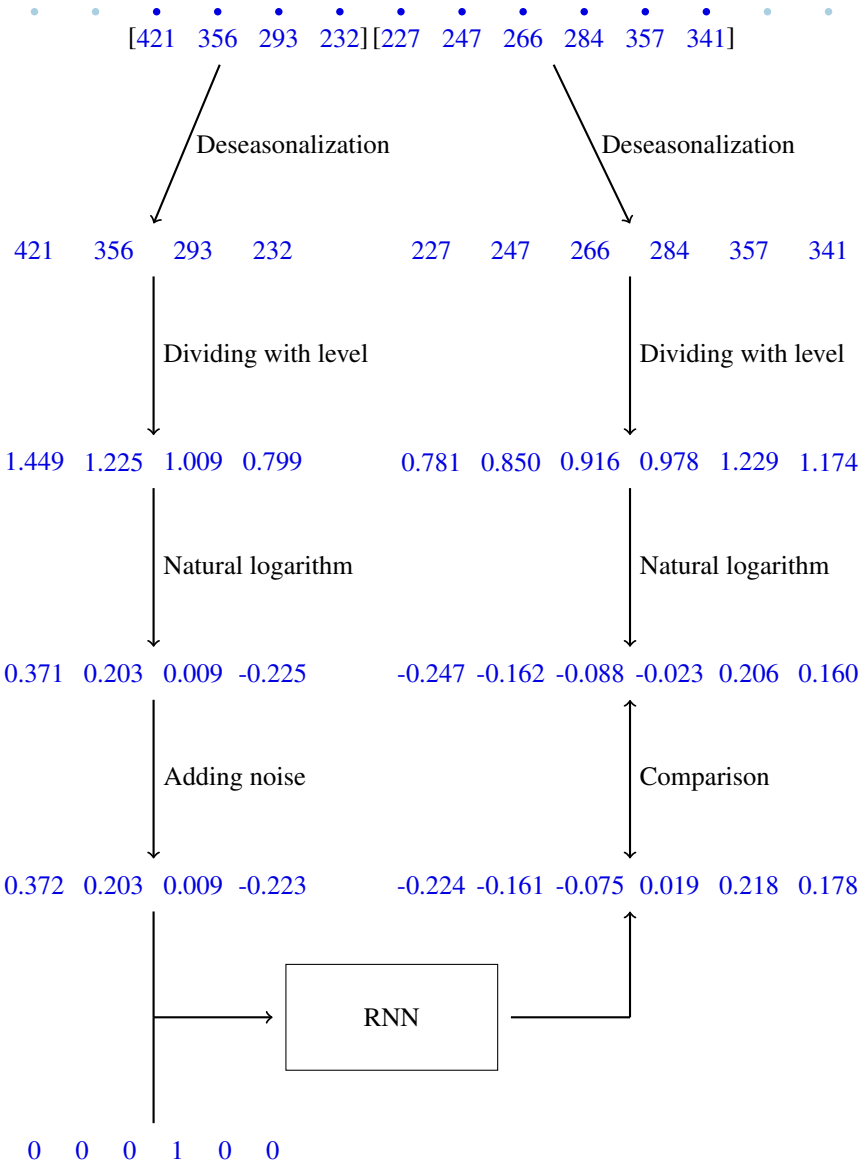


Figure 2.12: The values from the input window and the output window are preprocessed separately. After being preprocessed, the values from the input window are concatenated with a one-hot encoding of the origin of the series. The concatenated vector is input to an RNN. The output from the RNN is compared to the preprocessed values from the output window.

0.5 allows for the possibility to adjust for a model that tends to have a positive or negative bias. If the model tends to have a positive bias, we can set the value of τ to a value less than 0.5, for example 0.48. A model that predicts $\hat{y} = 12$ when $y = 10$ would usually have a loss of 2. With the pinball loss, the loss is $2 * (10 - 12) * (0.48 - 1) = 2.08$. Hence, we give a larger penalty when the model predicts too large values, and correspondingly a smaller penalty when the model predicts values that are too low. The pinball loss for each output node is averaged over all the output nodes to gain the pinball loss for the specific run through the network. The pinball loss for all the runs of the different windows through a network is averaged over the different runs.

In order to obtain a network's forecast for the next values in a series, the last values in the series are preprocessed and concatenated with the one-hot encoding of the origin and sent through the network. The output vector of the network is re-leveled and re-seasonalized by multiplying with the level and seasonality components to achieve the predicted values. The final results are decided by a combination of the top N networks for each series, where N is three or four depending on the resolution. These are the networks with the smallest pinball loss for a series during training.

2.3.2 M4metalearning

The second best contribution to the M4 competition was the method *M4metalearning*, which achieved an OWA of 0.838. The method is written in `R`. The model combines nine different well-known forecasting algorithms. A model is trained to assign weights to the different methods. A linear combination of the nine forecasts from the simple methods is used as the final result.

Firstly, the training set is prepared by splitting the training values in the M4 competition into a training set and a test set as shown in figure 2.13. It is done by splitting up the original training set *M4-train* into a new training set *M4metalearning-train* and a new test set *M4metalearning-test*. The last *h* observations in the training set is assigned to a test set, where *h* is the number of steps required in the forecasting horizon from the competition, and is hence also the length of the original test set *M4-test*. The remaining values from the original training set is assigned to the new training set *M4metalearning-train*.

The nine simple forecasting are all from the *forecast* package of `R`:

1. ARIMA like described in 2.1.7 using the method `auto.arima(stepwise=FALSE, approximation=FALSE)`
2. ETS like described in 2.1.8 using the method `ets()`
3. NNETAR - a feed-forward neural network where a single hidden layer is fitted to the lags as described by Hyndman (2017) using the method `nnetar()`
4. TBATS - the ES state space trigonometric, Box-Cox transformation, ARMA errors, Trend and Seasonal components model like described by De Livera et al. (2011)

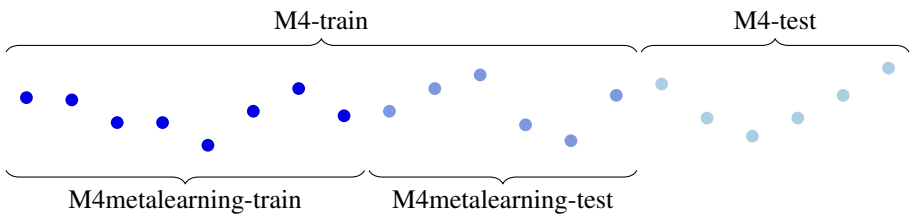


Figure 2.13: The M4metalearning method preprocesses time series before training a model. The training part *M4-train* of a time series from the M4 competition dataset could be divided into a training set *M4metalearning-train* and a test set *M4-metalearning-test*, so that the test-set *M4-test* is left out for testing.

using the method `tbats()`

5. STLM-AR - using the method `stlm(modelFunction=stats::ar)`
6. Random walk with drift like described in 2.1.6 using the method `rwf(drift=TRUE)`
7. The theta method like described in Assimakopoulos and Nikolopoulos (2000) using the method `thetaf()`
8. The naïve method like described in 2.1.4 using the method `naive()`
9. The seasonal naïve method like described in 2.1.4 using the method `snaive()`

For each time series, M4metalearning-train is used as input to all nine of the basic forecasting methods. This results in nine forecasts $f_1, f_2, ...f_9$ of length h that has the ground truth *M4metalearning-test*. In addition, for every series a set of 42 features are calculated. The features are labeled a. These includes for instance the length of the series, the number of seasonal periods in the series and the number of times the series crosses the median of the values.

An XGBoost model is trained on all series using these 42 features as input. The output of the model is a vector of length nine. The output of XGBoost is labeled $y(a)$. The ith entry is then labeled $y(a)_i$ That is one entry for each of the basic methods. When the model is finished training, the entries are supposed to indicate how good each basic method is doing for a time series with 42 specific features. To make the entries sum up to 1, the softmax function is used on the vector. The model is trained to minimize the loss function

$$L_{\text{OWA}}(y, a, f, \textit{M4metalearning-test}) = \sum_{i=1}^{M} \frac{e^{y(a)_i}}{\sum e^{y(a)}} \text{OWA}(f_i, \textit{M4metalearning-test}).$$

Finally, when the model is trained, each of the original series can be forecasted. The 42 features are extracted from a series. These are input to the XGBoost model which decides how large weight that should be assigned to which of the nine basic forecasting methods. A forecast is created using the nine basic forecasting methods and the weights are used to create a final forecast for submission.

	N	Horizon	Aggregation function	Weight
Yearly	3	6	Exponential weights with base 0.3	$scores^{-2}$
Quarterly	8	8	Mean	$scores^{-2}$
Monthly	10	18	Mean	$scores^{-2}$
Weekly	13	13	Mean	$exp(scores^{-1})$
Daily	8	14	Exponential weights with base 0.5	$scores^{-2}$
Hourly	24	48	Mean	$scores^{-1}$

Table 2.4: The number N of observations from the last part of the training set that is used as a hold-out for estimating the quality of the different forecasting methods.

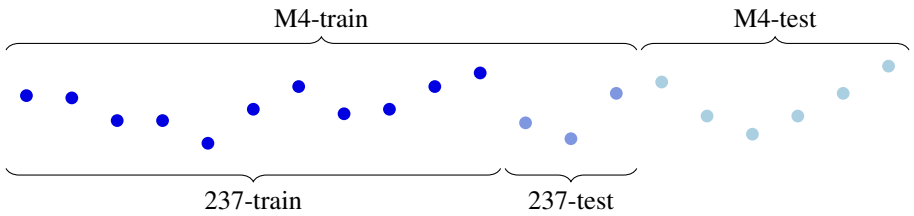


Figure 2.14: The WESM method preprocesses a yearly time series before training a model. The values from the original training set are labeled *M4-train* and the values from the original test set are labeled *M4-test*. The values from the original test set cannot be used for training. Therefore, the original training set values are split into a new training set *237-train* and a new test set *237-test*. The length of the new test set depends on the resolution of the series.

2.3.3 WESM

The third best contribution to the M4 competition was the method *Weighted Ensemble of Statistical Models* (WESM), which achieved an OWA of 0.841. The method is written in `R`. The model assigns weights to different forecasting methods for each series and the final result is a weighted average of standard forecasting techniques.

For each series, a set of standard forecasting methods $m_1, m_2...m_i$ are chosen based on the resolution of the series and if the series has a significant trend or seasonality. For instance, for a quarterly series with no trend and no seasonality five methods are used: the naïve method, SES, the theta method, the optimized theta method and ETS. The training set *M4-train* into a new training set *237-train* and a new test set *237-test*. The last *N* observations in *M4-train* is assigned to a test set, where *N* is smaller than or equal to the forecasting horizon. The value of N can be seen in table 2.4. For instance, N is set to 3 for yearly series which is less than the forecasting horizon of 6, and N is set to 8 for quarterly series which is the same as the forecasting horizon. An example of how a training set is divided can be seen in figure 2.14.

The remaining values from the original training set is assigned to the new training set *237-train*. The chosen methods are used on *237-train* to predict *237-test* and the sMAPE

between the forecasts and the real values are calculated so that each method $m_1, m_2, ...m_i$ has a set of sMAPE scores for each predicted value. The sMAPE scores the predicted values in the horizon for one method are combined using an aggregation function given by table 2.4. The aggregation function is a weighted average between the sMAPE scores. Usually, equal weights are used so that the aggregation function is simply the mean of the scores. For yearly and daily series, however, the scores from the most recent observations count the most and the weight decreases exponentially for each timestep further away (earlier) in time. This results in one score for each of the forecasting methods, $s_1, s_2...s_i$.

The scores are translated to weights. The methods to perform this transformation depends again on the resolution to the time series. This gives the weights $w_1, w_2, ...w_i$. The weight for method k is given by

$$w_k = \begin{cases} \frac{1}{s_k^2}, & \text{if resolution is yearly, quarterly, monthly or daily} \\ \frac{1}{s_k}, & \text{if resolution is hourly} \\ e^{\frac{1}{s_k}}, & \text{if resolution is weekly} \end{cases}$$

Finally, the set of methods $m_1, m_2...m_i$ takes *M4-train* as input and predicts the forecasting horizon required in the M4 competition. Each forecast is multiplied with its associated weight to generate the final forecast for the series. That is, the final prediction for each step in the forecasting horizon will be

$$\hat{Y} = \sum_{k=1}^{i} w_k * \hat{Y}_k,$$

where w_k is the weight for method k, $\hat{Y}_k$ is the prediction made by method k for the value Y and i is the number of different forecasting methods considered for this series.

2.3.4 GROEC

The fifth best contribution to the M4 competition was the method *Generalized Rolling Origin Evaluation Combination* (GROEC), which achieved an OWA of 0.843. The algorithm is written in `R`. The method simply combines four different forecasting methods with weights proportional to the quality of the cross-validation result of each.

Four individual forecasts

Unlike ES-RNN, GROEC is a local method. Each time series is treated for itself. First, the forecast for a time series is calculated using each of the four algorithms: DOTM, OTM,

ETS, and ARIMA. Each method is used both to find a forecast for the unknown horizon as required by the M4 competition, but also to forecast the last path of the training set so that cross-validation can be used to estimate the quality of each algorithm. A short description of each of the four methods follows.

The Theta model created interest with researchers after being the most accurate in the M3 competition (Makriadis and Hibon, 2000). The original method is described by Assimakopoulos and Nikolopoulos (2000). Fioruci et al. (2015) later describes the Optimized Theta Method (OTM). The concept is based on modifying the local curvatures of a time series. This change is obtained by a coefficient, called the Theta-coefficient. First, the linear regression of the original time series is found, as shown in figure 2.15. The point where the linear regression intersects with the y-axis is labeled α and the slope is labeled β. Then $Y_1, Y_2, ...Y_n$ is transformed to a *Theta-line* such that $Y_t(\theta) = \theta * Y_t + (1-\theta) * (\alpha + \beta * t)$. Figure 2.16 shows the results of transforming a time series with different values of θ. A θ-value smaller than 1 makes the curves of the time series smaller. In the extreme case where $\theta = 0$, the time series becomes linear. A θ-value larger than 1 makes the curves larger.

In the Theta method, the original time series is decomposed into two or more theta lines. The Theta lines can be regarded as individual time series and are extrapolated individually using a simple forecasting method such as for example ES. The Theta-model used in the M3 competition used a Theta-line with $\theta = 0$ and one with $\theta = 2$. The first theta line is linear and the forecast was found by extending the line. The second theta line was predicted with SES. In this way, the first theta line predicts the long-term behavior of the time series, while the second theta line approximates the short-time behavior. For each point in the forecasting horizon, the average of the different forecasts is found. This is the prediction for the original time series. The time series is also deseasonalized and before the whole process and the re-seasonalized after the process.

The optimized theta model distinguishes from the original theta model in the way that it optimizes the theta parameters by finding theta values that give a low error on the last part of the known time series. When combining the forecasts, the weights are not necessarily equal. More details can be found in Fioruci et al. (2015).

Barker (2019) describes the Dynamic Optimized Theta Model (DOTM). Easily explained, in this model, α and β are individual for each timestep. The details of the procedure are left out of this thesis. In GROEC, the method `dotm()` from the *forecTheta* package was used.

ETS is described in section 2.1.8. In the GROEC-method, the method `ets()` from the *forecast* package was used. Different ETS-methods are tried and the best one is chosen for the forecast.

The ARIMA method is described in section 2.1.7. In the GROEC method, The method `auto.arima()` from the *forecast* package of `R` was used. This method chooses appropriate values for p, d and q.

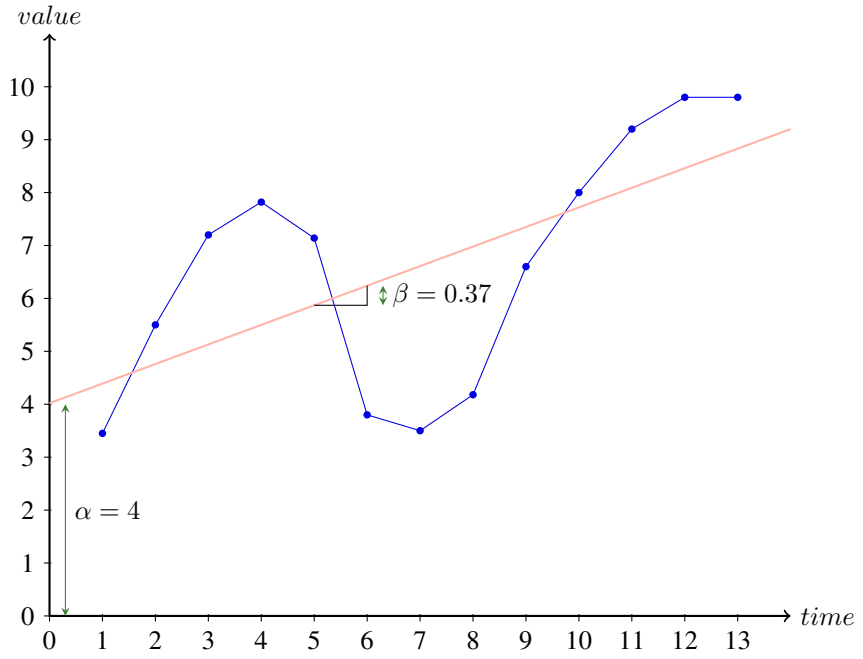


Figure 2.15: The values α and β that are used in the theta method. Least Squares Regression is used to find the line that best fits the data points. α is the y-coordinate where the regression line intersects with the y-axis. β is the slope of the regression line.

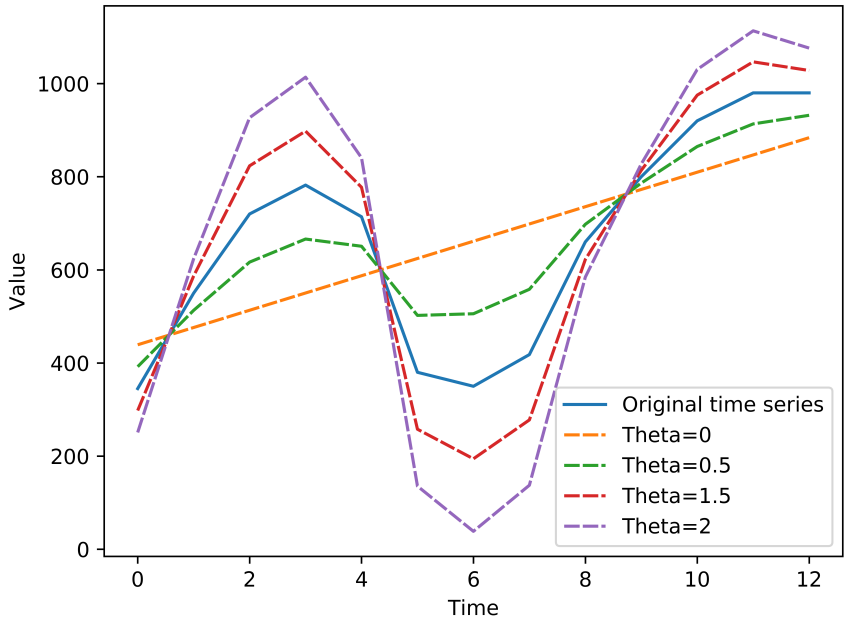


Figure 2.16: The blue time series is transformed to different Theta-lines using the formula $Y_t(\theta) = \theta * Y_t + (1-\theta) * (\alpha + \beta * t)$, where α and β are derived from linear regression. The larger the θ, the more extreme become the curves. A θ less than 1, however, makes the curves smaller than original.

Cross-validation

The novel thing about GROEC is not the four methods described so far, but the way they are combined to create the final forecast. Let t be the number of timesteps in the training set for a time series. Let H be the number of steps in the forecasting horizon. Then n is the number of steps in the training and test set together. There are six different origins, denoted by n_1, n_2, ..., n_6. The first origin is defined by the equation

$$n_1 = \begin{cases} n - H, & \text{if } n - H \geq 5 \\ 5, & \text{otherwise} \end{cases}$$

The rest of the origins are decided by the number $m = \lfloor \frac{H}{6} \rfloor$. We have

$$n_i = n_{i-1} + m \qquad \text{for } i < 1.$$

The GROE loss function is calculated for each individual method.

$$\text{GROE} = \sum_{i=1}^{6} \sum_{j=1}^{min(H, n-n_i)} g(y_{n_i+j}, \hat{y}_{n_i+j}),$$

where $g(y, \hat{y})$ is the error function:

$$g(y, \hat{y}) = 0.5 * \frac{\text{sAPE}(y, \hat{y})}{\text{sMAPE*}} + 0.5 * \frac{\text{ASE}(y, \hat{y})}{\text{MASE*}}.$$

Here, sMAPE* and MASE* is the sMAPE error and the MASE error, that the naïve2 method would have done in this case. The formula corresponds to the OWA score in the M4 competition. Usually, the OWA is done over whole series

Combining the Forecasts

A score is calculated for each of the four methods, as the inverse of the loss:

$$s_i = \frac{1}{l_i}$$

A low loss is good, and will then give a high score. The weight for a method is defined as the score divided by the total score for all the methods:

$$w_i = \frac{s_i}{\sum_{j=1}^{4} s_j}$$

The four forecasts are combined using the weights.

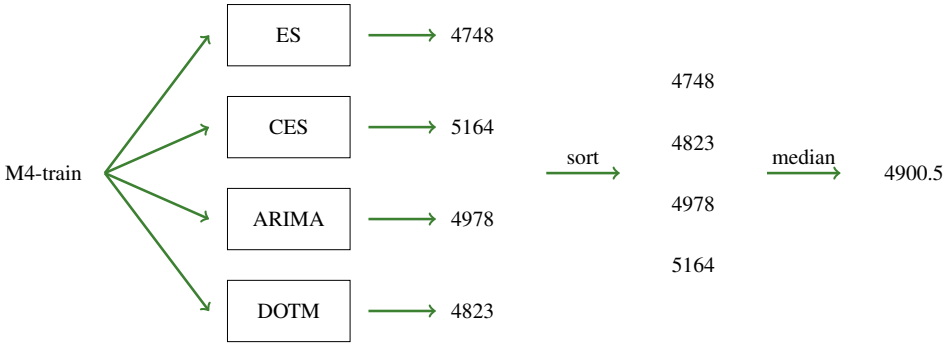


Figure 2.17: An example of how SCUM combines the forecast of four methods. A time series is forecasted by the four different methods ES, CES, ARIMA, and DOTM. For a given step in the horizon, they predict one value each. The predicted values are sorted and the median is found, which also is the average of the two middle values. That is the prediction that SCUM will make for that step in the horizon.

2.3.5 SCUM

The 6th best contribution to the M4 competition was the method *Simple Combination of Univariate Models* (SCUM), which achieved an OWA of 0.848. The method is written in `R` and combines four different forecasting methods and use the median of the results as the final forecast.

Four Individual Forecasts

Each time series is forecasted using each of the four algorithms: ES, complex ES, automatic ARIMA and DOTM. ES is described in section 2.1.4. For yearly, quarterly, monthly and daily series, the `ets()` function from the *forecast* package was used. For weekly and hourly series, the `es()` function of the *smooth* package is used. Complex ES method is described by Svetunkov and Kourentzes (2016). `auto.ces()` from the *smooth* package was used. The ARIMA method is described in section 2.1.7. `auto.arima()` function of the *forecast* package was used. DOTM is described in section 2.3.4. In SCUM, the `dotm()` method of the *forecTheta* package was used.

Combining the Forecasts

The forecasted value is the median of the four produced forecasts. For example, if ES forecasts 4748, CES 5164, ARIMA 4978, and the dynamic optimized theta method 4823, then the final forecast will be 4900.5. Figure 2.17 shows how SCUM combines the individual forecasts.

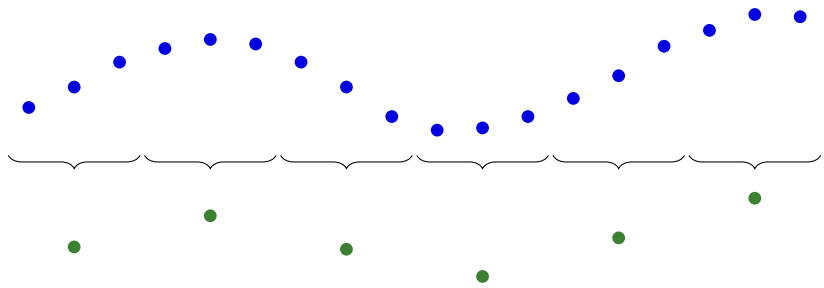


Figure 2.18: A monthly series (blue) with frequency $m = 12$ is aggregated into groups of $k = 3$. The sum of three values in a group becomes the new value of the aggregated series (green). The aggregated series has a frequency of $\lfloor \frac{m}{k} \rfloor = \lfloor \frac{12}{3} \rfloor = 4$.

2.3.6 THIEF Combination

The 7th best contribution to the M4 competition was the method *Temporal Hierarchical Forecasting Combination* (THIEF Combination), which achieved an OWA of 0.860. The method was written in `R`.

Temporal Hierarchical Forecasting

Temporal hierarchical forecasting (THIEF) is a technique for seasonal time series used in combination with regular forecasting algorithms. In this approach, a time series y is preprocessed by *aggregating* it, meaning that groups of subsequent observations are combined into one single value. An example of this is shown in figure 2.18. The observations are simply combined by summation. The number of observations that are aggregated k must be a factor in the frequency of the series m. For instance, a monthly series with seasonality $m = 12$ can be aggregated into groups of 2, 3, 4, 6, or 12. This results in another series $y**[k]$ with fewer observations than y. The frequency of the aggregated series will be $\frac{m}{k}$. For example, a monthly series aggregated into groups of 3 will have a frequency of 4. The aggregated series can be forecasted individually and the predicted values could be split up to give a forecast for the original series. By aggregating a series using different values of k, several forecasts can be created that can be combined into a single forecast for the original series. See Athanasopoulos et al. (2017) for details about the procedure.

THIEF Combination

THIEF Combination calculates two different forecasts using THIEF. One forecast is used with ARIMA as the base function for forecasting the aggregated series and one with the Theta method as the base function. This is accomplished using the `thief()` function from the *thief* package in `R` passing the argument `usemodel` equal to `"arima"` and `"theta"`. The average of the two forecasts is used as the final forecast for THIEF Com-

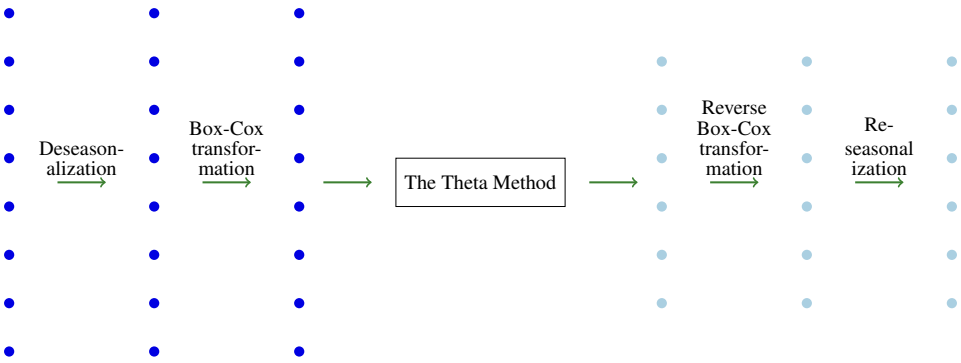


Figure 2.19: The lifetime of a time series being forecasted with Theta Box-Cox. The series is first deseasonalized if it has a seasonal pattern. Then Box-Cox transformation is applied. Then the Theta method is used for forecasting the time series. The predicted values are reverse transformed with Box-Cox and re-seasonalized if it was earlier deseasonalized.

bination.

2.3.7 Theta Box-Cox

The 8th best contribution to the M4 competition was the method *Theta Box-Cox*, which achieved an OWA of 0.861. The algorithm is written in `R`.

Figure 2.19 shows the process a time series goes through using the Theta method. The seasonal component of quarterly, monthly, and hourly data is checked for seasonality. In the case that there is a seasonal pattern, the time series is deseasonalized using multiplicative decomposition. Then, Box-Cox transformation (Box and Cox, 1964) is applied to all series. Put simply, this is a way to transform non-normal dependent variables into a normal shape. Every observation y_t is translated to w_t using the formula:

$$w_t = \begin{cases} \ln y_t, & \text{if } \gamma = 0 \\ \dfrac{y_t^{\gamma} - 1}{\gamma}, & \text{otherwise} \end{cases}$$

A value of γ is chosen such that the seasonal variation is about the same across the whole series. After deseasonalization and Box-Cox transformation, the series is ready to be forecasted. This is done with the theta method. The function `thetaf()` from the *forecast* package is used. Reverse Box-Cox transformation is done with the formula

$$y_t = \begin{cases} e^{w_t}, & \text{if } \gamma = 0 \\ (\gamma w_t + 1)^{\frac{1}{\gamma}}, & \text{otherwise} \end{cases}$$

using the same gamma values that were used for transformation. Finally, the predictions are re-seasonalized.

2.3.8 Predilab

The 10th best contribution to the M4 competition was the method *Predilab*, which achieved an OWA of 0.869. The algorithm is written in `R`. A different approach is used for different kinds of resolutions.

For yearly and quarterly series the method Theta4-ARMA is used. No textual description of the method is given except that it is a generalization of the theta method. It has not been prioritized to study the details of this method.

For a series that has a monthly, weekly or daily resolution, the training set is divided in two like in figure 2.13. We name the first part of the series *predilab-train* and the second part *predilab-test*. All the seven following methods are used on predilab-train to give a forecast for predilab-test:

- Seasonal naïve
- Naïve2
- SES
- Holt
- Damped
- Theta4
- Theta4-ARMA (a variation of Theta4)

The sMAPE scores of the methods are calculated. The three best methods are used on M4-train to give a forecast for M4-test. The three predictions are averaged.

Hourly series is tested on whether they have a weekly frequency. That is a repeating pattern every 24 * 7 = 168th step. If a time series has a weekly frequency, seasonal naïve with frequency 168 is used. Otherwise, seasonal naïve with frequency 24 is used.

2.4 Reproducibility

What is meant by reproducibility is not clearly defined. Stodden (2011) distinguishes between replication and reproduction. She defines replication as the process of rerunning the code used by the original researcher on the same data obtaining the same result. Reproduction, on the other hand, is a broader term. It includes both replication and the process of obtaining equivalent findings with some independence from the original code or data. Bollen et al. (2015), on the other hand, refer to replicability as the process of duplicating results using the same procedures, but different data than in the original study. They refer to reproducibility if both the same procedures and materials are used. Here, replication is the broader term.

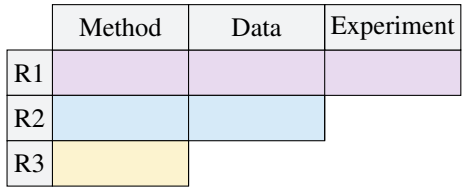


Figure 2.20: The reproducibility level of a computer science experiment is decided by its documentation.

2.4.1 A Reproducibility Framework

Due to conflicting definitions and the lack of a common standard, Gundersen and Kjensmo (2018) provided a framework for evaluating reproducibility in empirical AI research. They define reproducibility in empirical research as "*the ability of an* independent research team *to produce the same results using the same* AI method *based on the* documentation *made by the original research team*". The individual team should use the same AI method, but that does not mean that they have to use the same implementation of the method. The result captured by the performance measures should be the same as in the original experiment for the experiment to be reproducible.

They distinguish between three different degrees of reproducibility. The reproducibility degree of an experiment is decided by the information contained in the experiment's documentation, as shown in figure 2.20. For an experiment to be R3 reproducible, also called method reproducible, the method used by the original research team must be documented. For instance, they could report that they used k-means for solving a clustering problem. For an experiment to be R2 reproducible, or data reproducible, the documentation must not only include information about the method used, but also information about the data used. The dataset should be available. What data that is used for training, validation, and testing should be reported. In order to achieve R1 reproducibility, experiment reproducible, the documentation must also contain information about the experiment that was conducted. That includes source code, the operating system used, and parameters used in the experiment. A full overview of information required for each of the three factors can be found in table A.1.

To put it differently, for an experiment to be reproducible at all, an individual research team has to reproduce the same results. When reproducing the experiment, the implementation and data used decide the reproducibility degree as shown in table 2.21. The results of an experiment are method reproducible when the execution of an alternative implementation of the AI method produces the same results when executed on different data. They are data reproducible if an alternative implementation, but the same data is used. The results are experiment reproducible if both the same method is used and the same data. The definition of replication in Stodden (2011) corresponds to this definition of experiment reproducible. Both the definition of method reproducible and the definition of data reproducible fit under the definition of reproducibility provided by Stodden (2011).

	Data	Implementation
R1	Same	Same
R2	Same	Different
R3	Different	Different

Figure 2.21: The results of an experiment have a reproducibility degree, which depends on whether the same implementation of the AI method is used and whether the same data is used when producing the same results.

There are positive and negative sides with both ends of this ”scale”. An experiment documented to an R1 degree might be easier for an independent research team to reproduce. Also important is that having all these details from the original experiment increases the independent researchers' trust in the initial study's results. From the view of an independent research team, an experiment documented to an R1 degree might be preferable. However, more documentation work is required from the original research team. In addition, if an experiment could be reproduced using a different implementation and a different dataset, that is positive for the original research team. Their method does not only work with an exact implementation at a specific dataset, but is more general than that. As a result, the original research team may have little motivation for documenting more than the method.

Chapter 3

State of the Art

Like mentioned earlier, reproducibility in AI has received increased attention during the last years. Research includes AI tasks like language modeling, reinforcement learning, image classification, and recommending systems. A number of papers are collected and a summary is given for each of them.

3.1 On the State of the Art of Evaluation in Neural Language Models

Several novel recurrent neural network architectures have recently claimed to be the new state-of-the-art. Due to the usage of different codebases and limited computational resources, there existed no fair comparison of the models before Melis et al. (2017) conducted an experiment running a variety of recent published algorithms against each other and some standard recurrent neural network architectures.

Three different datasets were used. Two word-based datasets, and one character-based dataset. For each of the datasets, three standard recurrent architectures are trained: LSTM, RHN, and NAS. Three LSTM networks of different depths are trained, so that there are five different models. Hyperparameters are optimized by Google Vizier, a black-box hyperparameter tuner. Each of the five models is given a parameter budget of 10000 and 20000 parameters, which is the total number of trainable parameters in the model. As a result, ten different models were trained.

The models were compared against the results of recently published ”better” versions of LSTMs, RHNs, and NASs. On the two word-based datasets, the recently published architectures were outperformed by the standard methods. For the character-based dataset,

the novel algorithms were not outperformed. The authors blame the number of training epochs, which is only about one-tenth of the number used in some of the novel methods. The conclusion is that standard LSTM architectures today still outperform more recent models when properly regularised.

3.2 Are GANs Created Equal? A Large-Scale Study

Lucic et al. (2018) examined different Generative Adversarial Networks (GAN) models to find out if any of the algorithms performed better than others.

A GAN is a subclass of generative models. The task of generative models is to learn patterns in data and generate new data that could just as well have been drawn from the original dataset. For example, we could create a model that is trained on a bunch of real human faces to create new images of new human faces that no one has seen before. A GAN is a neural network that uses a generative model (generator) together with a discriminator model (discriminator) to become good at creating new examples that look like they belong to the distribution. While the generator learns to generate new data, the discriminator learns to discriminate between real and fake data. In other words, the discriminator is a classifier that takes as input either a real or a fake data example and learns to classify it as either of the categories. The generator can be trained to generate better examples by making it generate examples that are more confusing for the discriminator. In this way, a GAN can be viewed as a two-player game between the generator and the discriminator. While the discriminator is getting better discriminating between real and fake data, the generator becomes a better data generator (Walia, 2017).

Several different versions of GANs has recently been proposed, and up to this point in time, there was no clear consensus on which models performed better than others. One reason for this is the lack of a robust and consistent metric to measure which GAN that performs better. You can imagine several different models that enable to generate human face images. How can you decide which images that looks most like real people? The authors discuss several evaluation metrics and provide a fair and comprehensive comparison of state-of-the-art GANs. A proposed evaluation metric is *Fréchet Inception Distance*, *FID*. It is computed by considering the difference in the embedding of true and fake data. The generated data are embedded into a feature space given by a specific layer of Inception Net. Viewing the embedding layer as a continuous multivariate Gaussian. The Fréchet distance between the two Gaussians is used to quantify the quality of the samples.

Four popular datasets from GAN literature were chosen - CelebA, Cifar10, Fashion-Mnist and Mnist. First, a wide one-shot setup with 100 samples of hyper-parameters were selected. Then a narrow two-shots setup with 50 samples from more narrow ranges were selected manually after the wide range search. For each five epochs in the hyperparameter optimization, the best *FID* between 10k samples generated by the model and 10k samples from the test set were computed. In the end, the best *FID* across the training run was chosen. There was no algorithm that clearly outperforms the others.

It is concluded that most models can reach similar scores with enough hyperparameter optimization and random restarts. There was not found any evidence that any of the tested algorithms consistently outperformed the original GAN algorithm. The authors claim that it is necessary to report a summary of the distribution of results, rather than the best result achieved. It is suggested that future GAN research should be based on more systematic and objective evaluation procedures.

3.3 Deep Reinforcement Learning that Matters

Reinforcement learning differs from supervised learning in that it does not have any labeled training data to learn from. Instead, an *agent* is interacting with its environment and learns what is good and bad actions by achieving rewards or penalties. The agent learns by trial and error. The goal is to maximize the reward.

Henderson et al. (2017) address the problem of literature reporting a wide variety of results for the same reinforcement learning algorithms. They experiment with four model-free policy gradient algorithms that are frequently used as baselines for comparison against novel methods. Two different environments are used.

To see how the choice of network architecture affects the performance of a model, they are varying the number of hidden layers and the number of nodes in each layer for all the models in both environments. Also the hyperparameters and activation function is varied. The results show that how good a model is, depends on the architecture. Which architecture that is the best is a matter of model and environment. An instance of a model that works best in one environment is not necessarily the best in the other environment. Thus, the architecture must be adjusted to the specific environment the agent is learning in. Hyperparameter agnostic algorithms are suggested. That is, algorithms that itself is doing the hyperparameter optimization to ensure that some good hyperparameters for a specific problem. For a baseline algorithm, an architecture and hyperparameters should be found such that the performance of the model matches the performance of the original baseline algorithm.

Reward rescaling is to multiply the reward generated from an environment by some scalar for training. The researchers train one of the models using several reward scales. The reward scale does not always affect the result, but sometimes it has a large effect. They warn against using reward scaling and refers to a more principled approach.

Due to the major concern in machine learning regarding randomness during the learning phase, an experiment is being conducted in which the goal is to find out if the average of a number of networks trained with different random seeds was a good measure. 10 networks are trained with all parameters fixed except for a random seed. The networks are divided into two groups of five, and it turns out that the two groups can be very different. No rule of thumb is given when it comes to the number of runs with different random seeds, but it is recommended that there are many.

In order to tell how the environment properties affect variability in algorithm performance, the models are trained on the same four environments. It varies which models that outperform the others. Consequently, it is possible to make an algorithm seem better by reporting only the environments in which a proposed novel algorithm outperforms some baselines, or simply to be lucky if you only test in one environment. The authors recommend that a wide range of environments should be tested and reported, also the ones where the novel model was outperformed.

Lastly, the researchers are testing if commonly used baseline implementations of the same model are comparable. It is not unlikely that authors implement their own versions of baseline algorithms to compare against. For that reason, an experiment is conducted where several implementations are tested against each other. The results show that implementation differences can have dramatic impacts on performance. For this reason, it is important that also the baseline code is made available.

3.4 Unreproducible Research is Reproducible

Bouthillier et al. (2019) extends the works by Melis et al. (2017) and Lucic et al. (2018) to image classification. In contrast to NLP and GANs, the evaluation metric in image classification is simple. An image belongs to a specific class and you can measure what percentage of the images that are classified correctly. If an experiment is not reproducible, it must be something different with the model, not with the evaluation metric. In contrast to the RL experiments in Henderson et al. (2017), the environment is strongly controlled and there is only a small number of sources to random variations in the algorithm. If an image classification experiment is not reproducible, we cannot blame those variations.

Depending on how much information that is used when repeating an experiment, reproducibility is divided into three different categories: methods reproducibility, results reproducibility, and inferential reproducibility. We have methods reproducibility if reusing the original code leads to the same results. If we reimplement the experimental setup and achieve statistically similar results, we have results reproducibility. Inferential reproducibility is the concept that a finding or a conclusion is reproducible if one can draw it from a different experimental setup, for instance using a different dataset.

10 popular deep learning models for image classification were trained several times. For each model, 10 different seeds were used for initialization of the model parameters and ordering of the data presented by the data iterator. All models were trained twice on six different datasets: one in a *biased scenario* where the hyperparameters were the same for all models (those that were best for one model) and one in an *unbiased scenario* where the hyperparameters were optimized given a specific budget. It is found that which model that performs best varies with the random seed used. This applies particularly for simpler datasets. To ensure inferential reproducibility, one should train several models with different seeds on various datasets. One algorithm is rarely the best on all datasets, but in that case, also the negative results should be reported.

3.5 Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommender Approaches

Dacrema et al. (2019) is conducting a systematic analysis of deep learning based proposals for top-n recommendation tasks. That is, algorithms that for a given user, recommend a number, *n*, of items that the user is likely to be interested in. It could be movies, goods in an online store or other things. Now that methods are published continuously, it is difficult to keep track of the current state-of-the-art methods, and for that reason, the authors analyze the degree of reproducibility within the field.

Papers that were published on the conferences KDD, SIGIR, TheWebConf (WWW), and RecSys between 2015 and 2018 were manually scanned. A paper was considered relevant if it proposed a deep learning based technique to solve the top-n recommendation problem. 18 relevant papers were found. A paper was categorized as *reproducible* if the source code and at least one dataset used in the original paper was available. If not available, the original authors were contacted, and if they could provide code and data, the paper would still be categorized as reproducible. It was found that seven out of 18 papers met this definition of reproducible. Furthermore, it is attempted, for these seven papers, to run the source code against some baselines to check whether they perform better than the baselines. Seven baselines were picked out and fine-tuned for each paper's algorithm and dataset. Only one of the seven models was able to outperform all the baselines.

The authors conclude that reproducing published research is challenging. The reasons why many articles might not be reproducible are many: the source code is not shared; the code used for hyper-parameter optimization, evaluation, data preprocessing or baselines are not shared; data is not shared; or the computational complexity to reproduce an experiment could be high. They claim that a *phantom progress* is happening in the field of recommendation approaches. Many new approaches are published, but are not necessarily good. Novel algorithms are compared against baselines that are not certainly a strong baseline or that are not properly fine-tuned. It is suggested that future research practices are more accurate with respect to the evaluation of algorithmic contributions.

3.6 Objectivity, Reproducibility and Replicability in Forecasting Research

Makridakis et al. (2018) tried to reproduce an algorithm using machine learning to achieve a high degree of accuracy. Several points needed clarifying, but they did not succeed in getting in touch with the authors of the article. Furthermore, they tried to replicate the results of a paper that used eight machine learning methods on the M3 competition data. The results where approximately replicated. They wrote a paper about the previous find-

ings to *Neural Networks*, stating that all statistical forecasting methods were more accurate than machine learning methods. The paper was rejected with the reason editors claiming that many machine learning methods are superior to statistical models. Makridakis et al. (2018) could not find any such results and the editors would not give any references on this. The paper was however accepted by *Neurocomputing* and to *PLOS ONE*. Frustrated after being ignored by both the authors of the paper they tried to replicate and the editors of *Neural Networks*, they suggest how the field could work towards reproducibility and replicability. They have got nine concrete suggestions:

1. Forecasting journals should require all information needed to reproduce or replicate an experiment
2. The dataset used for evaluating a proposed method should be publicly available
3. Describe Methods: Any method implemented within the paper should be clearly defined
4. The authors should specify what software was used for producing the published results, as well as its version.
5. The authors should define the formulas used for evaluating their results, including all the measures, metrics and criteria exploited
6. Forecasting journals should offer the readers the ability to comment on the papers published, submitting thoughts, questions and requirements publicly visible
7. Editors of forecasting journals should regularly call for replications of important studies and encourage reviewers to accept such submissions on the basis of the quality of the experiments conducted, the significance of the hypotheses tested and the new information provided
8. Editors and reviewers should be objective, judging the work made by the authors based on its quality and potential impact and not just on the conclusions drawn
9. New methods should be tested on multiple large-sized datasets

The authors are organizing the M4 competition. To ensure reproducibility and replicability, the participants will be required to publish their code as well as a detailed description of the method used.

Chapter 4

Proposed Methodology

In order to answer the research questions, an attempt will be made to reproduce some of the methods from the M4 competition. Here, the word *reproduce* is used as in the definition of reproducibility provided by Gundersen and Kjensmo (2018). As a researcher, independent from the original research teams in the M4 competition, I will attempt to produce the same results using the same AI methods as the original teams, based only on their documentation. Using the original implementation and dataset, it will be examined if the methods are of reproducibility degree R1, experiment reproducible. Some methods are machine learning models. Those could either be reproduced by training a new model from scratch or by using a pre-trained model to forecast the series. In this project, all such models will be trained from scratch.

4.1 Methods to Reproduce

A requirement for the methods to be attempted reproduced is of course that they are publicly available. 36 out of the 49 participants have made their source code publicly available, some with instructions on how the code could be reproduced. For it to be relevant to run the code, it must not only be a skeleton code, but it needs to be fully runnable or require only small changes.

Makridakis et al. (2019) has already evaluated the reproducibility of most of the methods. This is valuable when selecting which methods to attempt to reproduce. The result can be seen in table B.1. Based on the sMAPE difference between the original method's result and the result of the retrained method, each model is categorized as *fully replicable* (in the original paper *full replicable*), *replicable* or *not replicable*. The running time for a majority of the models is given.

Priority	Id	Name of Method
1	118	ES-RNN
2	245	M4metalearning
3	237	WESM
4	72	forecaster18
5	69	GROEC
6	36	SCUM
7	78	THIEF Combination
8	260	Theta BoxCox
9	238	Card
10	39	Predilab
11	5	4Theta
12	251	DOTM
13	250	Tartu M4 Combination

Table 4.1: The methods that will be attempted reproduced in prioritized order.

Based on (Makridakis et al., 2019)'s result, some methods are chosen to be attempted reproduced in this project. The chosen methods are listed in table 4.1. First of all, a method must be publicly available in the official GitHub repository for the M4 competition to be considered. Methods categorized as "not replicable" by Makridakis et al. (2019) will be skipped. The same goes for methods that are marked with a running time of more than two months. The last condition for a model to be rerun is that it must be better than all of the benchmark methods from the M4 team. We are not interested in the reproducibility of methods that are not performing well. The best benchmark method is the theta method. Methods having a score worse than this (OWA larger than 0.897) will not be attempted rerun.

Regarding the time limit for this project, it would be an optimistic goal to run the code of all 13 methods. For that reason, priority must be given to the methods that have the lowest OWA scores. How many methods that will be investigated will depend on how much time this takes.

4.2 Rerunning the Methods

Starting at the top of table 4.1, the following procedure will be applied for each method: Based on the documentation given by the original researchers, an environment will be attempted set up in a Docker image, ready for running the source code. The Docker image is downloaded to two different computers. Five instances of the docker image is created on each computer and a single rerun of the algorithm is performed in each container to produce forecasts for the time series. They are not necessarily run at the same time, but if the computer resources are sufficient, several reruns of the same or different methods can

be carried out at once. If the code is unnecessarily difficult to run or has a running time of more than two months, the method will be considered not feasible to reproduce.

4.2.1 Documentation by the Original Researchers

Each participating team was asked to deliver the source code used for generating the forecasts and instructions on how to exactly reproduce the submitted forecasts. These deliveries are located in the team's folder in the M4 competition's official GitHub repository. The team's GitHub folder makes up the documentation that this experiment will be based upon.

4.2.2 Docker Image

Based on the submitted documentation a Docker image is created. The docker image sets up an environment for the method to run in. This makes it easy to run the same algorithm several times in the same environment, also on different computers. Besides, it makes this work easily reproducible.

4.2.3 Computers

When the Docker image is ready, it is downloaded to computer A and computer B. Both computers run the Ubuntu 18.04.4 LTS. Computer A has more cores and working memory than a usual home computer. It has a CPU with 28 physical and 56 logical kernels of the type Intel Xeon Gold 6132. The total working memory was 755 GiB. Computer B is a more typical home computer. It has a CPU with 4 physical and 8 logical kernels. The total working memory was 15.6 GiB.

4.3 Evaluating a the Results

According to Gundersen and Kjensmo (2018), for an experiment to be reproducible, the independent researchers must produce the *same result*. However, *same result* is not defined. In the context of the M4 competition, this could mean that the same forecasts should be produced when rerunning a method. On the other hand, two models could predict different values that are equally good according to some performance measure. Different forecasts might have the same OWA value in the M4 competition. Therefore, it might be just as interesting to investigate if the *scores* of a rerun is the same as the score of the original submission. As a *score*, we could use the same performance measure as in the competition, namely OWA. One last possibility is to compare the OWA values for several methods

from the competition at once. When rerunning them and giving them new scores, does that change the overall order of the best-performing methods?

If a rerun of a method is feasible, then five reruns of that method will be performed on computer A and five reruns will be performed on computer B. For each rerun we will obtain a result consisting of 100,000 forecasts with different horizon lengths. The following subsections describe how the reruns will be evaluated.

4.3.1 Similarity Between the Forecasts

For each rerun, the result will be compared to the original method's result. Both a rerun and the original submission will consist of 100,000 forecasts. Each series' forecast will consist of several forecasted values, one value for each step in the horizon. For instance, a yearly series would have six forecasted values in a rerun and six forecasted values from the original submission. Each of those forecasted values in a rerun are compared to the corresponding forecasted value from the original submission. The comparison is done with sAPE. This will create a schema of the same format as a rerun, but instead of containing forecasts, containing the sAPE value between the rerun forecasted values and the original forecasted values.

The sAPE schema is created for all ten reruns. An average sAPE schema is created where each value is the average of all corresponding values from the five schemas. The average sAPE schema is then processed the following way: For each of the six different time series resolutions (yearly, quarterly, and so on), the average sAPE is calculated for each step in the horizon. For example, the average is taken over all sAPEs for the first step in the horizon for yearly series, then the average is taken over all sAPEs for the second step in the horizon for yearly series and so on. This way it will be detected if the reruns are similar to the original submission early in the horizon or late in the horizon.

The sAPE values for a time series' horizon is then averaged, so that the sMAPE value for the series is obtained. The sMAPE values for all series in a rerun is calculated. Then the average is taken over the sMAPE averages for five reruns of the same method on the same computer. The value is compared to other methods and the same method on the other computer.

Next up, the proportion of the time series that have an sMAPE equal to zero is calculated. This share is averaged over the five reruns for a method on one computer. The same thing is done with all sMAPE equal to or below the threshold $1 * 10^{-5} = 0.00001$. This is such a small sMAPE value that the forecasts can be considered equal.

4.3.2 Similarity in the Performance

The previous subsection described a procedure to tell how equal the forecasts produced by the reruns are to the original forecasts. However, it does not say if the reruns are doing

better or worse than the original result. A well-performing method produces forecasts similar to values that are later observed in the future. These are the values from the test set. Hence, we can use some performance measure comparing the reruns to the test set and the original submission to the test set and see if they are doing equally good.

We can compare a forecast to the test set using OWA, like they also do in the competition for comparing the methods. The following is procedure is carried out:

1. The following is done for each rerun and for the original result:
   (a) The sMAPE and the MASE for each series is calculated as in the equations 2.8 and 2.9
   (b) The sMAPE and MASE is averaged over all series with the same resolution and origin
   (c) The sMAPE and MASE is averaged over all series having the same resolution
   (d) The sMAPE and MASE is averaged over all series having the same origin
   (e) The sMAPE and MASE is averaged over all series
   (f) The OWA is calculated for each of the previously mentioned categories as in equation 2.10
2. The OWA value over the five reruns for each of the mentioned categories
3. The average of the reruns are compared to the original OWA values. Is it better, worse or the same?

4.3.3 Variance Between the Reruns

When we run the same algorithm five times on the same computer, do we get the same result every time? And if we do not, how much does the result variate from run to run? To measure how different the forecasts are we could have simply used the variance or the standard deviation. However, these do not take into account the magnitude of the numbers. For example, we could measure the height of a child once every year in inches and then predict the height of the child next year. We run the same algorithm five times and calculate the values 44, 44, 45, 46, and 46. The standard deviation between the values is 1. If we have the same values in centimeters we would have got the values 111.8, 111.8, 114.3, 116.8, and 116.8 which has the standard deviation 2.5. The values, which in practice means the same, gives us different standard deviation. This becomes an issue when we have time series from different domains measuring totally different things. To solve the problem, we divide the standard deviation with the mean of the values. This is called the *coefficient of variation*. In this example, we will get the same coefficient of variation, namely 0.022 regardless of which unit is used for measuring.

The time series in the M4 competition training set do not all have the same magnitude. Figure 4.1 illustrates a histogram over the average value of all observed values for a series.

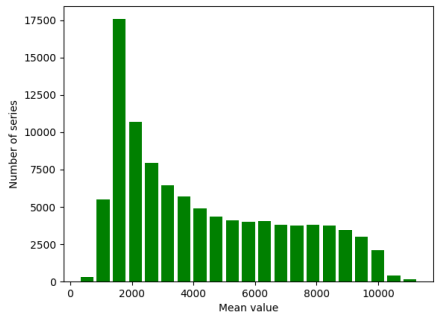


Figure 4.1: The time series varies in size. The figure illustrates the time series having a mean lower than 10511. There are 85 additional series that have larger means and are not included in the table.

Even though many series have a mean observed value of just under 2000, there are also series that have a mean observed value of around 10 000. 85 series is also left out of the histogram because they had extremely high means. H57 has the very largest values with a mean of approximately 525 000. Because of the large spread in magnitude of the series, the coefficient of variation is used to measure the variation of each method on each computer:

$$CV = \frac{\sigma}{\bar{x}},$$

where σ is the sample standard deviation between the values given by the formula

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})}{N-1}},$$

where $x_1, x_2 ... x_N$ is forecasted values for the same timestep of the same series, forecasted with the same method on different reruns, $\bar{x}$ is the mean of those values and N is the number of values. The reason for using the sample standard deviation (using $N-1$ in the denominator) and not the population standard deviation (using N in the denominator) is that we have been running a method five times out of infinite many times. We only have a sample of five different reruns.

After finding the coefficient of variation for every forecasted value, we can take the average over each step in the horizon for all resolutions.

4.3.4 Difference Between Computers

Do we get the same results when we run the algorithm on two different computers? For every method, we have five reruns on each of the two different computers. That means that for every value in the forecasting horizon there are five forecasted values from one computer and five forecasted values from another computer. If all ten values are the same,

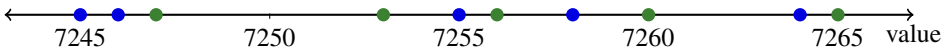


Figure 4.2: The figure illustrates two samples of numbers, green and blue. It is difficult to say if they are picked from the same distribution or not.

the answer is easy: yes, they are equal. But what if the method is not deterministic and predicts different values each time and we got ten different values? Figure 4.2 illustrates this case. The values are all different, but all in the range 7245 to 7265. The blue values have a mean of 7253.6 and the green values have a mean of 7256.2, so the blue values seem to be a bit smaller than the green ones. However, the difference between 7253.6 and 7253.2 is small compared to the magnitude of the numbers. When putting a number on how equal the samples are, there are some requirements for this metric:

1. A larger distance between the two groups should give a larger error
2. If we multiply all the numbers in the two groups with some constant, the error between the two new groups should be the same as for the old ones
3. If we add a positive constant to all the numbers in the two groups, that should give a smaller error (assuming that all numbers were already positive)
4. A larger variation in a group should give a smaller error

Using the example groups from table 2.4, we can take a deeper look into the requirements. The letters $f...j$ can be thought of as different algorithms run five times each on computer A and B. For instance, algorithm f produces the values 10, 11, 12, 13 and 14 on computer A and the values 11, 12, 13, 14 and 15 on computer B. This might be forecasted values for one timestep in a horizon for one series in the M4 dataset. We can refer to the groups of numbers as f_A and f_B. Requirement 1 says that a larger distance between the two groups should give a larger error. g_A is equal to f_A, while g_B is higher than the values in f_B. There is a larger difference between the g-groups than the f-groups. Hence, the error between the g-groups should be larger than the errors between the f-groups. When it comes to requirement 2, we can compare the f-groups to the h-groups. All values in the h-groups equals to the values in the f-group times 10. The f-groups and the h-groups should have equal errors, so that the error scales with the magnitude of the different series. All values in the i-groups equals to the values in the f-groups added to 100. This brings us to the third requirement. The difference between 110 and 111 should count less than the difference between 10 and 11. Lastly, we have the j-groups, which equals to the f-groups except that the variation between j_A is larger than the variation between the numbers in f_A. This should give a smaller error, since a large variation within the groups would make it more likely that the two groups are from the same distribution.

A metric that matches the requirements is *difference relative to mean and standard deviation* (DRMSD):

$$DRMSD = \frac{(\mu_A - \mu_B)^2}{\mu * \sigma},$$

	f		g		h		i		j	
	A	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
$\mathbf{x}_1$	10	11	10	12	100	110	110	111	6	11
$\mathbf{x}_2$	11	12	11	13	110	120	111	112	9	12
$\mathbf{x}_3$	12	13	12	14	120	130	112	113	12	13
$\mathbf{x}_4$	13	14	13	15	130	140	113	114	15	14
$\mathbf{x}_5$	14	15	14	16	140	150	114	115	18	15

Table 4.2: Examples of values that could be produced when the algorithms f, g, h, i, and j are running on computer A and B.

where μ_A and μ_B is the average of computer A and B, respectively; μ is the average of the averages (also the average of all the numbers); and σ is the average of the sample standard deviation of computer A and the sample standard deviation of computer B. If the mean of the averages are equal, then $DRMSD = 0$. There is no limit on how large the DRMSD can be, as there is no value for how large the difference can be between the two groups of numbers. The metric is undefined if the average of the averages or the average of the standard deviations is zero. This means that if an algorithm is deterministic, the error is never defined. The algorithm could be deterministic independent from the computers such that it always predicts the same value. It can also be deterministic such that it gives five equal values on computer A and five other values on computer B which are equal to each other, but unequal to the values from computer A. In both cases we can use the *percentage difference* as the error metric:

$$PD = \frac{|\mu_A - \mu_B|}{\mu} * 100\%$$

DRMSD will be calculated for all forecasted values in the horizon. A graph over the average DRMSD for each timestep sorted on resolution will be created. The average DRMSD for all forecasted values in the horizon for each series will be calculated. The series will be sorted on resolutions and origins and the average DRMSD within each group will be calculated.

Chapter 5

Results

The docker images can be found at Docker Hub (Soleim, 2020a) together with a step by step instruction for how you can rerun the algorithms. The Dockerfiles used for building the Docker images are available in the GitHub repository (Soleim, 2020b).

The top ten methods from the priority list (118, 245, 237, 72, 69, 36, 78, 260, 238, and 39) were attempted reproduced. The last three methods from the priority list (5, 251, 250) were not attempted rerun due to time constraints and limited resources in the project. Out of the ten methods that were attempted reproduced, eight (118, 245, 237, 69, 36, 78, 260, and 39) was possible to rerun. Four of these (118, 36, 260, and 39) were successfully rerun on both of the computers mentioned in 4.2.3. The other four (245, 237, 69, and 78) were only rerun on computer A, the one with the most resources. Two models (72 and 238) were considered infeasible to get up and running during this project. Table 5.1 shows an overview of this. The produced forecasts can be found in the GitHub repository (Soleim, 2020b).

5.1 The Rerunning of the Methods

A short description of the process of examining each method's source code and setting up the environment follows.

5.1.1 ES-RNN

ES-RNN was written in `C++` and was the only one of the ten examined methods that were not written in `R`. The author claims that the algorithm can be run on Windows, Linux, and

Method id	Name of method	Rerun on Computer A	Rerun on Computer B
118	ES-RNN	Yes	Yes
245	M4metalearning	Yes	Not enough working memory
237	WESM	Yes	Not enough working memory
72	forecaster18	Poorly documented	Poorly documented
69	GROEC	Yes	Not enough working memory
36	SCUM	Yes	Yes
78	THIEF Combination	Yes	Unknown error
260	Theta Box-Cox	Yes	Yes
238	Card	Poorly documented	Poorly documented
39	Predilab	Yes	Yes
5	4Theta	Not attempted	Not attempted
251	DOTM	Not attempted	Not attempted
250	Tartu M4 Combination	Not attempted	Not attempted

Table 5.1: An overview over which methods that were rerun on which computers.

macOS. The process of setting up this environment was a complicated process. A zip-file was submitted containing 43 different files in a folder structure. Even though some of those files were readme-files that explained a few things, it was not obvious where to start. The code uses DyNet, a neural network library developed by Carnegie Mellon University among others. Although it was time-consuming to get started with this method, it produced forecasts on both computer A and computer B. The forecasts were carried out one resolution type at a time. After running the main code in `R`, some code written in `C++` had to be run to produce the final forecast file for each resolution. No programmatic procedure was given for combining the forecast files for the different resolutions into one final submission file, so a `Python` script was created to accomplish this.

In the preliminary work for this thesis, it was detected one variable that most likely had changed between the training of the original model and the source code delivery. This was a variable indicating the number of seasonal patterns in the series. In the submitted source code, this was set to 0. A warning message showed up when running, indicating that something might be wrong with this variable. Additionally, there was a large difference between the original hourly forecasts and the forecast produced by a rerun. Due to these reasons, the variable was changed to 1 before rerunning the code.

5.1.2 M4metalearning

The authors of M4metalearning provides both a pre-trained model used for the submission in the competition and source code written in `R` for retraining the model. In this project, the latter one is of interest. A dedicated `R` package provides the core of the method. A Docker image is created from ubuntu 18:04, installing `R` and the necessary libraries. The method was able to run on computer A, but crashed after a few days of running. It was investigated whether the crash was caused by a specific time series. A binary search over all time series was executed and some specific time series turned out to be guilty: ”Y12146”, ”Y21168”,

”Y22801”, ”Q5619”, ”M16993” or ”D2085”. The source code was modified to forecast all time series except these. After the rerun, the forecast files were updated manually, adding the lines with the given ids and giving every value in the forecasting horizon the value ”NA”.

The method did not run on computer B. The code crashed shortly when trying to assigning the training set with an error message informing that a vector could not be allocated.

5.1.3 WESM

The authors specify that `R` version 3.5.0 was used and the *forecast* package version 8.2, so these were installed with the given versions. A demo file showing how the forecasts could be executed was attached, but the forecasts were only stored in a variable and were not saved to a file. A main file `run.R` was created to run the forecasts and write them to a file. This file was written such that forecasts for only one specific resolution was executed. The forecast files were later concatenated using the same `Python` program as for ES-RNN.

5.1.4 forecaster18

The code is split up into one `R` file for each resolution and a file for merging the results. The dataset is loaded as a local file with RData format. This file is not enclosed, neither is any code for transforming the original dataset provided by the organizers of the competition into RData format. Not having the data needed for running the code, this method was considered infeasible to reproduce in this project.

5.1.5 GROEC

No description of a procedure for how to reproduce the results of the GROEC method is provided. However, only a single file is delivered, which simply had to be run in `R` after installing the necessary packages, to produce forecasts for all the series. It is given in the source code that `R` version 3.4.3 was used.

5.1.6 SCUM

Two programs with `R` code are delivered. The first program transforms the dataset from the CSV-files provided by the competition's organizers into an RData format. The second program gives forecasts from the RData. This is explained in the method's description. It is stated that `R` version 3.4.3 was used, in addition to the version of all the libraries that are used. Also the output of the `R` function `sessionInfo()` is given. The source code is

written so that the user can easily decide exactly which time series from the dataset he/she want to forecast. All time series from the dataset were chosen to be forecasted in one run.

5.1.7 THIEF Combination

This method was already provided with a Dockerfile in the submission folder. A Docker image was simply built from this Dockerfile. The code did not require any changes to run. One file was produced for each resolution, which was saved one by one after all time series with the given resolution was forecasted. At computer A, two of the Docker containers stopped during running, one such that quarterly and yearly time series were not finished yet, and another such that only the yearly time series was left. The Docker containers had exited with error 137, an out of memory error. The code was edited to predict only the missing resolutions. This was considered reasonable to do to save computing time as this was understood to be a local method, where the prediction of one series is not affected by other time series. On computer B, there were larger problems running the code. After some days of running, it was noticed that the CPU usage was too low to be working on the forecasts. The Docker containers were still running and there was no error message in the nohup file. Yet, there was no further progression in the run according to the nohup files during the next days. After two weeks of trying forecasting different resolutions, the code was considered impossible to rerun on computer B.

5.1.8 Theta Box-Cox

Installing `R` and the necessary libraries to run the one delivered source code file was straight forward. Although the source code looked finished at first glance, it turned out after a run that for each new resolution to forecast, these new forecasts wrote over the old forecasts in the data frame. Some small changes in the source code solved the problem, so that it became possible to forecast all series in one run.

5.1.9 Card

This method is written in the programming language `Ox`. The dataset is meant to be stored in `in7`/`bn7` files, which is very fast for `Ox` to read. Unfortunately, these files were not available, neither was the code to generate them. The method was considered infeasible to reproduce.

5.1.10 Predilab

The submission consists of two source code files: one main file and one with auxiliary functions. There were no problems during the setup of the environment. Nevertheless,

when the forecasts were produced, the ids of the time series was not as expected. The original ids start with a letter giving the resolution of the series. Within that resolutions, all series are given a partial key. The combination of the letter and the partial key is the series' primary key or id. The ids are Y1, Y2, ..., Y23000, Q1, Q2, ..., Q24000, M1, and so on. The ids for the forecasts given by Predilab, however, are Y1, Y2, ..., Y23000, Q23001, Q23002, ..., Q47000, M47001, and so on. A `Python` script was created and run on the forecasts to give back their old id.

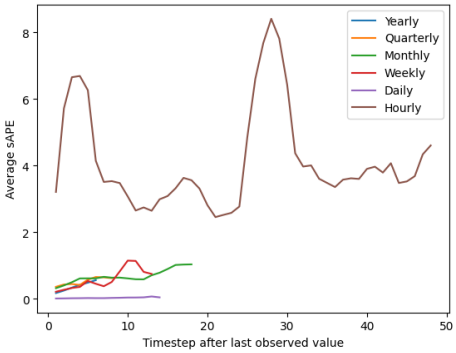
5.2 Similarity Between the Forecasts

For all the methods that were rerun, the sAPE for each step in the horizon averaged over the five reruns as described in section 4.3.1. The results from computer A is shown in figure 5.1. Notice that the y-axis differs from method to method. The methods are not supposed to be compared to each other in this figure. On the other hand, we can compare how a specific method did for the different resolutions. For instance, we observe that method 039 had a higher sAPE for daily series. It may look like the reruns predicted the other resolutions similarly to the original submission, while something might have been done differently with the daily series. The figure also illustrates that in some cases, values early in the horizon were predicted more similarly to the original submission than values late in the horizon. The corresponding results from computer B is shown in figure C.1 in the appendix. All the results from computer B are very similar to the results from computer A.

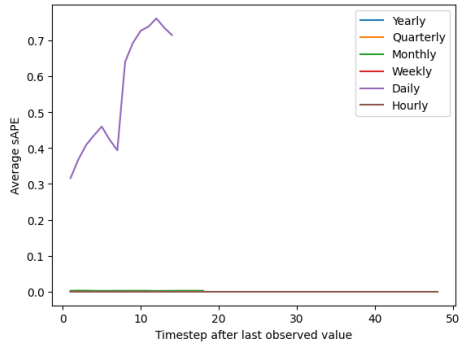
For each method, the average sMAPE is calculated for five reruns of the same methods on the same computer. The averages are shown in figure 5.2. We observe that method 118 has the highest sMAPE values, hence that is the method that gives the most different result compared to the original submission. Method 039 and 260 are the ones with the lowest sMAPE and are the methods that are most similar to their original submission.

Which proportion of the series that have an sMAPE equal to zero is shown in table 5.2. We observe that method 039 predict the same result as delivered in the original submission for 73% of the series, both using computer A and computer B. Method 078 also stands out, predicting the same result for 58% of the time series on computer A. This result differs from the result in figure 5.2 in that method 260 predicts no time series equally to the original submission, even though it has a low average sMAPE.

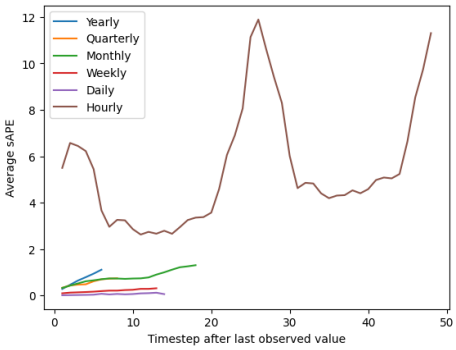
When increasing the threshold to $1 * 10^{-5}$ we get the results in table 5.3. Accepting all time series that have an sMAPE less than or equal to $1 * 10^{-5}$ as equal to the original submission, also method 237 and 260 predict the same values as the original submission for more than half of the series.



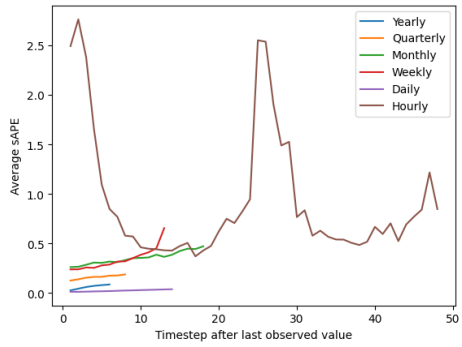
(a) 036-A



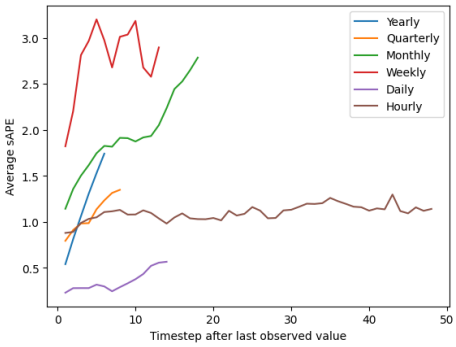
(b) 039-A



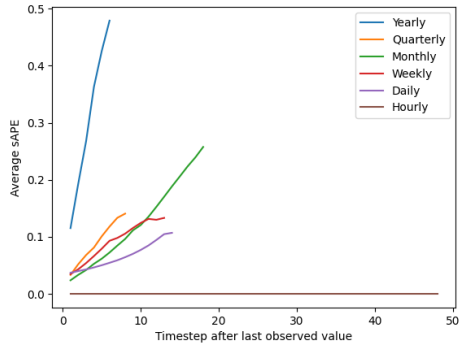
(c) 069-A



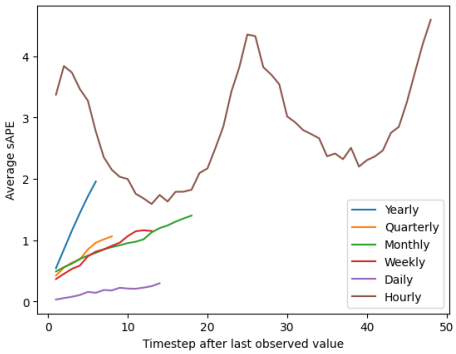
(d) 078-A



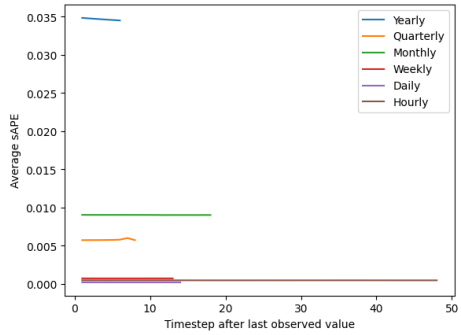
(e) 118-A



(f) 237-A



(g) 245-A



(h) 260-A

Figure 5.1: Average sAPE between the original submission and five reruns for each step in the horizon. The x-axis shows the timesteps in the forecasting horizon and the y-axis shows the sAPE.

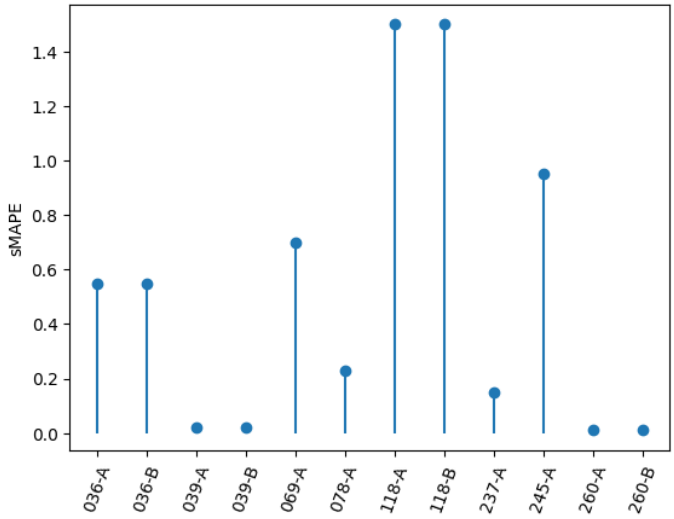


Figure 5.2: The sMAPE is calculated between all forecasted time series and the original submitted forecast computed with the same method. The sMAPE is averaged over all time series on the same rerun. Then the average is taken over the sMAPE value for the five reruns.

Method id	A	B
036	0	0
039	73	73
069	2	-
078	58	-
118	0	0
237	5	-
245	0	-
260	0	0

Table 5.2: The percentage of time series where the sMAPE between a rerun and the original submission was equal to 0 on computer A and B.

Method id	A	B
036	13	13
039	90	90
069	3	-
078	60	-
118	0	0
237	78	-
245	0	-
260	55	55

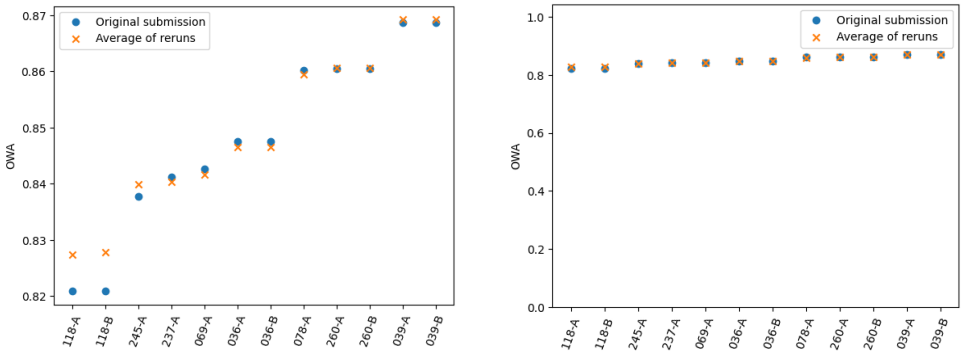
Table 5.3: The percentage of time series where the sMAPE between a rerun and the original submission was less than $1 * 10^{-5}$ on computer A and B.

5.3 Similarity in the Performance

The OWA is calculated as described in section 4.3.2 and compared to the original submission's OWA. The differences are shown in table C.1-C.12 in the appendix. Positive values show that the original submission had a greater OWA than the reruns, and hence that the reruns did better. Those are marked in green. Negative values show that the reruns did not do as good as the original submissions. Those are marked in red. Common for all the methods is that the reruns are doing a bit better in some categories and a bit worse in other categories.

A summary of the tables is shown in figure 5.3. It shows the OWA value for the original method compared to the average OWA value for the reruns of that method. 036, 069, 078, and 237 are overall doing a bit better on the reruns than the original submission. 039, 118, 245, and 260 are doing a bit worse on the reruns. However, the OWA values of the reruns do not change the order of the best methods in the competition.

Figure C.2 and figure C.3 also shows comparisons of the OWA values of the original method and the reruns, but for individual resolutions and origins. For some of the time series, the reruns do not provide the same order of the best methods as the original results. For example, for demographic series, method 118 was better than method 237 according to the OWA score. The reruns, however, gives the opposite result. It has to be said that the OWA values are very close and very small changes in the OWA values cause the change of order. Figure C.4 and C.5 shows the same results, but with the y-axis starting at zero. These figures illustrate both that the OWA values of the top 10 methods in the M4 competition are very similar, and they illustrate how close the rerun OWA values are to the original results.



(a) Truncated y-axis

(b) Y-axis starting at zero

Figure 5.3: Average OWA for the original submissions and for the reruns. The subfigures shows the same data, but in the first graph, the y-axis is cut off so that the details are available. The methods are sorted from the lowest original OWA to the highest original OWA.

Method id	Computer A	Computer B
036	0	0
039	0	0
069	0	-
078	0	-
118	0.00927	0.00941
237	0	-
245	0.01006	-
260	0	0

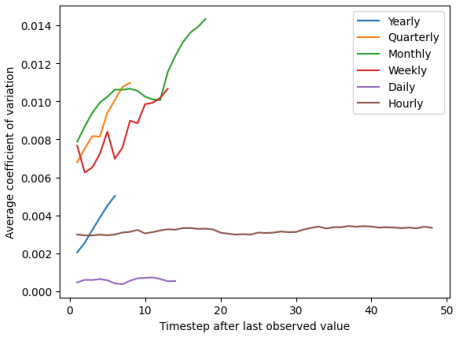
Table 5.4: Average coefficient of variation between five reruns of a method on the same computer.

5.4 Variation Between the Reruns

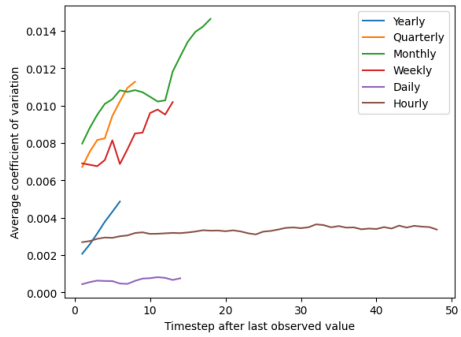
The average coefficient of variation between five reruns of a method on the same computer is shown in table 5.4. The methods 118 and 245 are the only ones that are non-deterministic. Figure 5.4 shows the coefficient of variation averaged over each timestep in the horizon for these methods.

5.5 Difference Between Computers

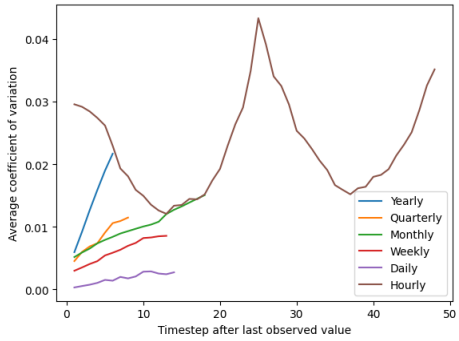
The methods that were rerun on computer B was 036, 039, 118, and 260. The DRMSD and the PD is shown in table 5.5. Method 118 was not deterministic. The DRMSD between the five reruns on computer A and computer B, for each step in the horizon, is shown in figure 5.5. Figure 5.6 shows the DRSMD for each resolution and origin.



(a) 118-A



(b) 118-B



(c) 245-A

Figure 5.4: Coefficient of variation between five reruns on the same computer.

	DRMSD	PD
036	-	0
039	-	0
118	0.00453	0.00508
260	-	0

Table 5.5: The average DRMSD and PD between all forecasted values from the five reruns on computer A and the five reruns on computer B.

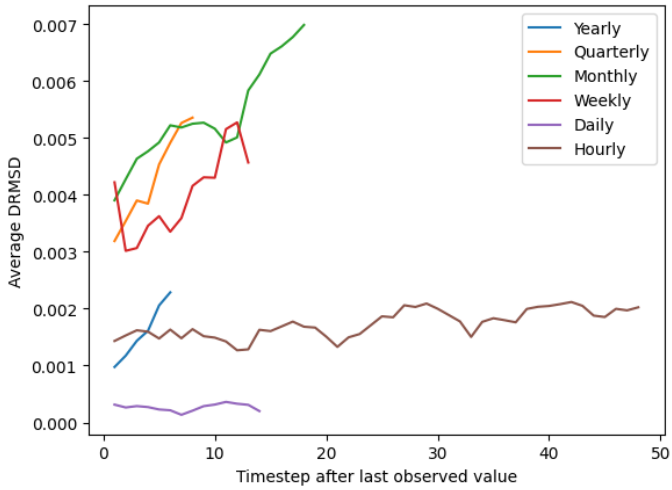


Figure 5.5: The DRMSD of the forecasted values between reruns of method 118 on two different computers.

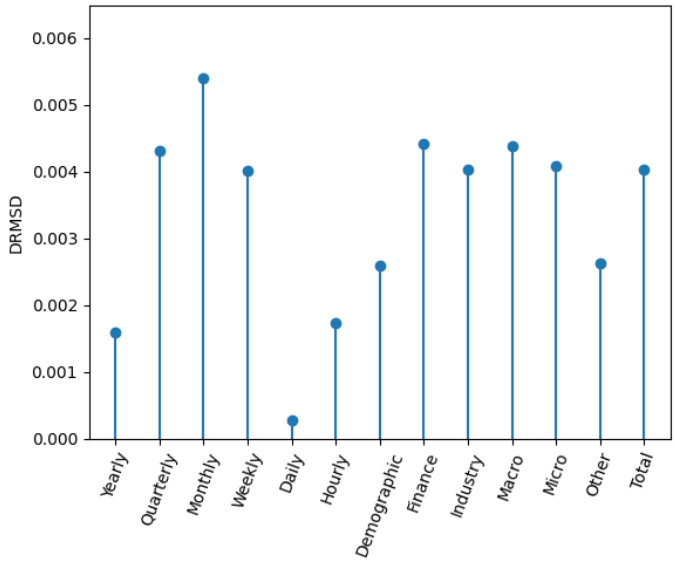


Figure 5.6: The DRMSD of the forecasted values between reruns of method 118 on two different computers.

Chapter 6

Discussion and Conclusion

This work aims to ensure that future research within the field of time series forecasting is reproducible. The three research questions mentioned in section 1.4 was:

1. To what degree is the top-performing methods in the M4 competition reproducible?
2. Which factors make research on time series forecasting difficult to reproduce?
3. How can we work for future research on time series forecasting to reach a higher level of reproducibility?

In this chapter, the research questions will be discussed and a conclusion will be made.

6.1 Discussion

6.1.1 To what degree is the top-performing methods in the M4 competition reproducible?

The experiments performed in this thesis is based on the definition of reproducibility by Gundersen and Kjensmo (2018): ”*Reproducibility in empirical AI research is the ability of an* independent research team *to produce the same results using the same* AI method *based on the* documentation *made by the original research team.*”. The meaning of *same result* is interpreted in different ways in this project.

Performing a Rerun

First of all, for a method to be reproducible, it must be possible to rerun the code. Most of the methods require small changes in the source code to be rerun. Eight out of the ten attempted reproduced methods were possible to rerun and at least get some forecasts. Two methods were not rerun with minor modifications, both due to missing information about the preprocessing of the dataset. The methods were not documented sufficiently enough for them to be experiment reproducible. To achieve this level of reproducibility, either the dataset should be given in the needed format, or the source code for transforming the original dataset should be given.

Similarity Between the Forecasts

None of the eight methods that were rerun produced the same forecast as the original submission for all time series. According to table 5.3, four of the methods produced the same forecasts for more than half of the time series.

Method 039 was the one that produced the same forecast for most of the series. It predicted the same values as the original submission for 90% of the series. Figure 5.1b shows that the daily series is predicted especially different in the reruns from the original submission. Method 039 has the same procedure for monthly, weekly, and daily series, so it is odd that only the results for daily series are different. An explanation for this might be that some variable for daily series, or some other code affecting the daily series, has been changed between the original run and the reruns.

The method with the second most similar forecasts to the original submission was the forecasts of method 237 with 78% of the forecasts equal to the original submission. Method 078 is number third with 60% and method 260 is the fourth with 55%.

These are the same four methods that comes best out of the scatterplot in figure 5.2. They are not only the methods that have the most time series predicted equally as the original submission, but they are also the methods that have on average most equal predictions to the test set according to sMAPE.

The reruns of methods 036, 069, 118, and 245 produced forecasts more different from the original submission. It must be said that even though the forecasts were not the same, they are not terribly different. The reruns of method 118 on computer B were the ones most different from the original submission with an average sMAPE of 1.66. To give understand what this value of sMAPE means, we can take an example. The sAPE between the values 1000 and 1017 is about the same. If the original method predicted the value 1000 for some point in the horizon, a rerun in this project could have predicted 1017. That is typically how different the forecasts of the reruns of method 118 are from the original submission.

In the description of method 245, it is mentioned that the results may slightly vary since one of the individual forecasting methods uses random initialization. The statement agrees

with these observations.

Similarity in the Performance

It could also be argued that the most important thing is not if the forecasts of a rerun is equal to the original forecasts, but whether the two forecasts are having a similar score according to some performance measure. The OWA of the original submission is compared to the average OWA of the reruns in section 5.3.

The result shows that the OWA is not always the same for the reruns as for the original submission. Nevertheless, they are proved to be very similar. All methods have some categories where it does better and some categories where it does worse. Half of the methods are doing a bit better overall and half of the methods are doing a bit worse.

Even though the final OWA score is not the same for any method, the results of the reruns did not change the order of the best methods in the competition. When we sort the methods according to OWA of the original submission and average OWA for the reruns, we get the same order. This is in spite of a tight competition with OWA scores all between 0.821 and 0.869 for the considered methods.

Variance Between the Reruns

The six methods 036, 039, 069, 078, 237, and 260 have no variation at all. All the forecasted values were the same in all five reruns. Those methods seem to be deterministic. 118 and 245 on the other hand, produced different forecasts on different reruns. Hence, they are non-deterministic. The variation is s slightly bit higher for method 245.

Figure 5.4 shows the coefficient of variation for every step in the horizon for every resolution. Method 118 has the highest variation in quarterly, monthly, and weekly series. For method 245, the hourly series is the ones with the highest variation. The coefficient of variation increases for forecasts further into the forecasting horizon.

Difference Between Computers

Three out of the six methods that were deterministic given the computer were rerun on both computers. All these produced the same forecasts on computer A as for computer B. Method 118 was the only non-deterministic method that was run on both computers.

The DRMSD of the forecasted values between reruns of method 118 on the two different computers is shown in figure 5.5. This figure looks very much the same as 5.4a, but with a different y-axis. The difference between the computers is proportional to the size of the variation between the individual reruns. There is nothing to suggest that the difference

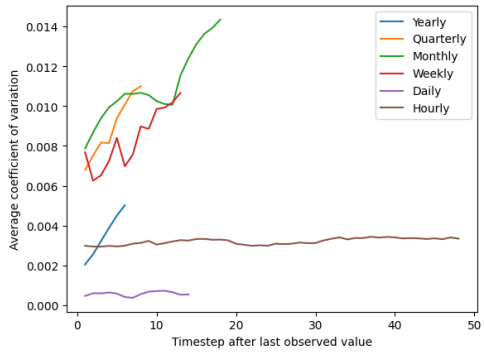


Figure 5.4a: Coefficient of variation between five reruns of method 118 on computer A. (repeated from page 62)

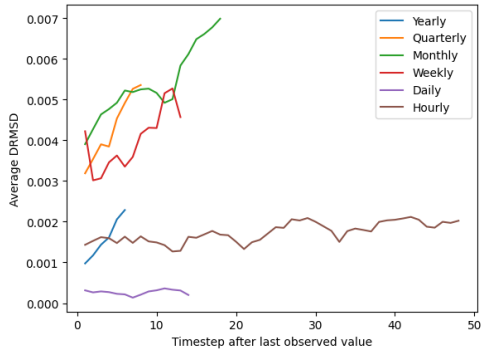


Figure 5.5: The DRMSD of the forecasted values between reruns of method 118 on two different computers. (repeated from page 63)

between the computers is larger than the difference between the individual reruns on the same computer.

6.1.2 Which factors make research on time series forecasting difficult to reproduce?

A quite obvious source to non-determinism among some of the methods was the use of randomness in the algorithm. Method 118 and 245 are both using randomness as part of their algorithms. Method 118 is using randomness several places in the algorithm. First, each series is assigned randomly to different networks for training. Prior to training, each network shuffles all its assigned series. When a series is preprocessed, noise is added to the series. Then, the smoothing coefficients are initialized uniformly between -0.5 and 0.5. Method 245 is a linear combination of nine different forecasting methods where one is a neural network initialized with random weights.

Randomness is a cornerstone in machine learning. Scardapane and Wang (2017) describes how important randomness is in neural networks. Randomness is part of the deal when choosing to use a machine learning model. Of course, a random seed can be saved so that others can use the same one later.

Threading can also cause non-determinism in the results. Lee (2006) shows how this can happen. A multi-core processor could have several cores execute different parts of the program at the same time. The result becomes timing sensitive.

Several of the methods behave deterministically, also when run on different computers. However, the forecasts are not equal to the forecasts submitted by the original researchers. The methods seem to be deterministic when run by the author of this thesis, but produced different results for the original authors of the methods.

One explanation may be that the code is unequal. There is no guarantee that the code delivered was not changed after running the methods. Section 5.1 describes how some methods contained small bugs that needed to be fixed for the code to run. It is not unlikely that some changes have happened when the participants in the competition has prepared their source code for delivery.

One reason may be that not the same version of the programming language or libraries is used. First of all, if a newer version of the programming language is used when a new researcher is trying to rerun some code than the version used by the original researchers, there might be functions that are removed such that the code is not runnable with the new version at all. The same might be the case if the researcher uses a too old version of the programming language. There might have come new functions or new syntaxes afterward that did not exist in that version. The latest version of `R` at the time of the deadline for the M4 competition was `R-3.5.0`. There have been eight updates since then and the newest version of `R` to date is `R-4.0.0`. An example of the difference between the two is the default value for the argument `stringsAsFactors` in the function `data.frame()`. In `R-3.5.0` this is set to `TRUE`, whereas in `R-4.0.0` this is set to `FALSE`. In the older version of `R`, the elements of a data frame were of class `factor`, while in the newer version they are of class `character`. This makes different functions available. This way, the exact same code can turn out different in different versions. `R-4.0.0` came out in 2020, so we know that this was not used in any of the deliveries. From the time at which the competition was announced in November 2017 until the delivery date in May 2018, there were two new `R` versions. Nor can we exclude that even older versions might have been used. Four out of the eight submissions had not specified which version of the programming language they were using.

Another possible explanation for the different results might be differences in hardware components. The hardware might have a significant impact on the results due to rounding errors in floating-point arithmetic according to (Hong et al., 2013).

Even though the code was run on two different computers, it was run in Docker containers from the same Docker image, simulating the same environment. The use of Docker image might have went against its purpose in this situation. It might be that the forecasts pro-

duced by the methods are different from computer to computer, but that the use of Docker provided deterministic results. Maybe the two computers had produced different results if the code had been run directly on the machines, not using a Docker image.

6.1.3 How can we work for future research on time series forecasting to reach a higher level of reproducibility?

When publishing any novel method, the researcher should ensure to document the environment in which the method has been running. That includes hardware, operating system, and the version of programming language and libraries used. To create a Docker image for others to run is an easy way to allow others to run the code in the same environment. A researcher should also ensure that the data is publicly available along with the source code. Any code for preprocessing the data should also be included. A seed could be used for random variables.

6.2 Conclusion

Ten methods were attempted rerun. Two of the methods were unable to predict any forecasts. Eight were able to predict forecasts for all time series in the M4 competition dataset on a computer with more cores and working memory than a regular home computer. Four of the methods were able to predict forecasts for all time series on a regular home computer. The forecasts produced by these two computers were similar. For the three methods that were deterministic when run on the first computer that were also run on the second computer, all of these were also deterministic on the second computer and predicted the same values as on the first computer. For the method that was non-deterministic on the first computer that was also run on the second computer, no evidence could be found that the two groups of forecasts were from different distributions. Hence, there was nothing that indicated that the two computers acted differently for any of the methods. Nevertheless, none of the methods gave the same forecasts as in the original submission for all time series. One can speculate about the reason for this. A main theory is that something with the environment in which the code was run was different from the original experiment. That could be the hardware on the computer where the method ran, the operating system, the programming language version or the version of packages used in the code. Even though the reruns not gave the same result as the original submissions, they did not change the order of the best methods in the competition.

6.3 Further Work

The methods analyzed in this project are specialized to give good results on the M4 dataset. The parameters has been fine-tuned manually, or by methods that are not reported, to give

the best result for this exact dataset. It is not reported how much resources were put into this process. For example, method 118 has different neural network architectures for the different resolutions. It would be interesting to test the algorithm on other datasets to see how they generalize. Method 237 uses a special technique for daily and hourly series where it finds highly correlated segments from other series in the training set and use that information to produce forecasts. Is this technique transferable to other datasets?

In this project, each method was tested in two different environments. The only parameter that distinguished the environments was the computer hardware. For the methods that acted deterministic on computer A, it was also the case that they gave the same result for computer B as for computer A. This experiment could be carried out in a larger scale, testing with even more computers. Will the algorithms from one Docker image always create the same result or can the hardware from the computer it is running on affect the results?

Another possibility is to create Docker images based on different operating systems and run them on the same hardware. In that way, one can examine if the environment in which a method is run can affect the result.

Bibliography

Aarts, A., Anderson, C., Anderson, J., van Assen, M., Attridge, P., Attwood, A., Baranski, E., 2016. Reproducibility project: psychology.

Assimakopoulos, V., Nikolopoulos, K., 2000. The theta model: a decomposition approach to forecasting. International Journal of Forecasting 16, 521–530.

Athanasopoulos, G., Hyndman, R.J., Kourentzes, N., Petropoulos, F., 2017. Forecasting with temporal hierarchies. European Journal of Operational Research 262, 60–74.

Barker, J., 2019. Machine learning in m4: What makes a good unstructured model? International Journal of Forecasting 36, 150–155.

Bollen, K., Cacioppo, J.T., Kaplan, R.M., Krosnick, J.A., Olds, J.L., 2015. Social, behavioral, and economic sciences perspectives on robust and reliable science.

Bouthillier, X., Laurent, C., Vincent, P., 2019. Unreproducible research is reproducible, in: Chaudhuri, K., Salakhutdinov, R. (Eds.), Proceedings of the 36th International Conference on Machine Learning, PMLR, Long Beach, California, USA. pp. 725–734. URL: http://proceedings.mlr.press/v97/bouthillier19a.html.

Box, G.E.P., Cox, D.R., 1964. An analysis of transformations. Journal of the Royal Statistical Society. Series B (Methodological) 26, 211–246.

Dacrema, M.F., Cremonesi, P., Jannach, D., 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. CoRR abs/1907.06902. URL: http://arxiv.org/abs/1907.06902, arXiv:1907.06902.

De Livera, A.M., Hyndman, R.J., Snyder, R.D., 2011. Forecasting time series with complex seasonal patterns using exponential smoothing. Journal of the American Statistical Association 106, 1513–1527.

Fioruci, J.A., Pellegrini, T.R., Louzada, F., Petropoulos, F., 2015. The optimised theta method. arXiv: Methodology .

Gundersen, O.E., 2019. Standing on the feet of giants — reproducibility in ai. AI Magazine 40, 9–23.

Gundersen, O.E., Kjensmo, S., 2018. State of the art: Reproducibility in artificial intelligence, in: AAAI.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D., 2017. Deep reinforcement learning that matters. CoRR abs/1709.06560. URL: http://arxiv.org/abs/1709.06560, arXiv:1709.06560.

Hong, S.Y., Koo, M.S., Jang, J., Kim, J.E.E., Park, H., Joh, M.S., Kang, J.H., Oh, T.J., 2013. An evaluation of the software system dependency of a global atmospheric model. Monthly Weather Review 11, 4165–4172.

Hyndman, R., Koehler, A., 2006. Another look at measures of forecast accuracy. International Journal of Forecasting 22, 679–688. doi:10.1016/j.ijforecast.2006.03.001.

Hyndman, R., Koehler, A., Ord, J., Snyder, R., 2008. Forecasting with exponential smoothing: the state space approach. Springer-Verlag.

Hyndman, R.J., 2017. Prediction intervals for nnetar models. URL: https://robjhyndman.com/hyndsight/nnetar-prediction-intervals/.

Hyndman, R.J., Athanasopoulos, G., 2018. Forecasting Principles and Practice. 2nd ed., OTexts: Melbourne, Australia. URL: OTexts.com/fpp2.

Lee, E.A., 2006. The problem with threads.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O., 2018. Are gans created equal? a large-scale study. NIPS'18 Proceedings of the 32nd International Conference on Neural Information Processing Systems 32, 698–707.

Lynnerup, N., N.L.H.R., Hallam, J., 2019. A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots.

M4Team, 2019. The m open forecasting center (mofc). URL: https://www.mcompetitions.unic.ac.cy/.

Makriadis, S., Hibon, M., 2000. The m3-competition: results, conclusions and implications.

Makridakis, S., 1993. Accuracy measures: theoretical and practical concerns. International Journal of Forecasting 9, 527–529. URL: https://EconPapers.repec.org/RePEc:eee:intfor:v:9:y:1993:i:4:p:527-529.

Makridakis, S., Assimakopoulos, V., Spiliotis, E., 2018. Objectivity, reproducibility and replicability in forecasting research. International Journal of Forecasting doi:10.1016/j.ijforecast.2018.05.001.

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2019. The m4 competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting 36. doi:10.1016/j.ijforecast.2019.04.014.

Melis, G., Dyer, C., Blunsom, P., 2017. On the state of the art of evaluation in neural language models. CoRR abs/1707.05589. URL: http://arxiv.org/abs/1707.05589, arXiv:1707.05589.

Niven, D.J., McCormick, T.J., Straus, S.E., Hemmelgarn, B.R., Jeffs, L., Barnes, T.R.M., Stelfox, H.T., 2018. Reproducibility of clinical research in critical care: a scoping review. BMC Medicine 16.

Pole, A., West, M., Harrison, J., 1994. Applied Bayesian Forecasting and Time Series Analysis. Taylor & Francis Group.

Scardapane, S., Wang, D., 2017. Randomness in neural networks: an overview. John Wiley Sons, Ltd 7.

Soleim, M., 2019. Reproducibility in time series forecasting. The preliminary work for this thesis.

Soleim, M., 2020a. Docker images for running the top m4 methods. URL: https://hub.docker.com/u/mariasoleim.

Soleim, M., 2020b. Reproduction of m4 methods. URL: https://github.com/MariaSoleim/M4-reproduction.

Stodden, V.C., 2011. Trust your science? open your data and code.

Stupple, A., Singerman, D., Celi, L.A., 2019. The reproducibility crisis in the age of digital medicine. npj Digital Medicine 2.

Svetunkov, I., Kourentzes, N., 2016. Complex exponential smoothing for seasonal time series.

Walia, A.S., 2017. Generative models and gans. URL: https://towardsdatascience.com/generative-models-and-gans-fe7efc20020b.

Appendices

Appendix A

Reproducibility

A.1 Variables that makes up the factors that decide the reproducibility degree

Factor	Variable	Description
Method	Problem	Is there an explicit mention of the problem the research seeks to solve?
	Objective	Is the research objective explicitly mentioned?
	Research method	Is there an explicit mention of the research method used (empirical, theoretical)?
	Research questions	Is there an explicit mention of the research question(s) addressed?
	Pseudocode	Is the AI method described using pseudocode?
	Hypothesis	Is there an explicit mention of the hypotheses being investigated?
	Prediction	Is there an explicit mention of predictions related to the hypothesis?
	Experiment setup	Are the variable settings shared, such as hyperparameters?
Data	Training data	Is the training set shared?
	Validation data	Is the validation set shared?
	Test data	Is the test set shared?
	Results	Are the relevant intermediate and final results output by the AI program shared?
Experiment	Method source code	Is the AI system code available open source?
	Experiment source code	Is the experiment code available open source?
	Software dependencies	Are software dependencies specified?
	Hardware	Is the hardware used for conducting the experiment specified?

Table A.1: The factors needed for differend degrees of reproducibility and the variables that specify them as specified by Gundersen (2019).

Appendix B

The M4 Competition

B.1 An overview of the M4 competition's contributions

Rank	Id	Replicability category	Code publicly available	Running time (min)
1	118	Fully replicable	Y	8056.0
2	245	Fully replicable	Y	46108.3
3	237	Fully replicable	Y	39654.8
4	72	Not considered	Y	-
5	69	Not considered	Y	-
6	36	Fully replicable	Y	4049.5
7	78	Fully replicable	Y	8575.0
8	260	Fully replicable	Y	25.0
9	238	Fully replicable	Y	2.1
10	39	Fully replicable	Y	6742.6
11	5	Fully replicable	Y	3335.9
12	132	Not replicable	Y	-
13	251	Fully replicable	Y	109.6
14	250	Not considered	Y	-
15	243	Not replicable	Y	-
16	235	Not replicable	Y	-
17	104	Unknown	Y	>2 months
18	223	Not replicable	Y	-
19	239	Not replicable	Y	-
20	211	Replicable	Y	232.1
21	231	Not considered	N	-
22	227	Fully replicable	Y	27432.3

23	82	Not considered	N	-
24	212	Not considered	N	-
25	236	Not considered	N	-
26	248	Not considered	N	-
27	30	Not considered	N	-
28	234	Not considered	N	-
29	24	Not considered	N	-
30	218	Unknown	Y	-
31	106	Fully replicable	Y	62242.9
32	43	Not replicable	Y	-
33	216	Not replicable	Y	-
34	169	Not replicable	Y	26719.2
35	241	Not considered	N	-
36	191	Not considered	N	-
37	126	Not replicable	Y	-
38	244	Not considered	N	-
39	70	Not considered	N	-
40	249	Not considered	N	-
41	252	Fully replicable	Y	393.5
42	255	Fully replicable	Y	154.8
43	9	Fully replicable	Y	63.6
44	256	Fully replicable	Y	72.5
45	253	Fully replicable	Y	37.2
46	91	Unknown	Y	-
47	219	Unknown	Y	-
48	225	Unknown	Y	-
49	258	Unknown	Y	-

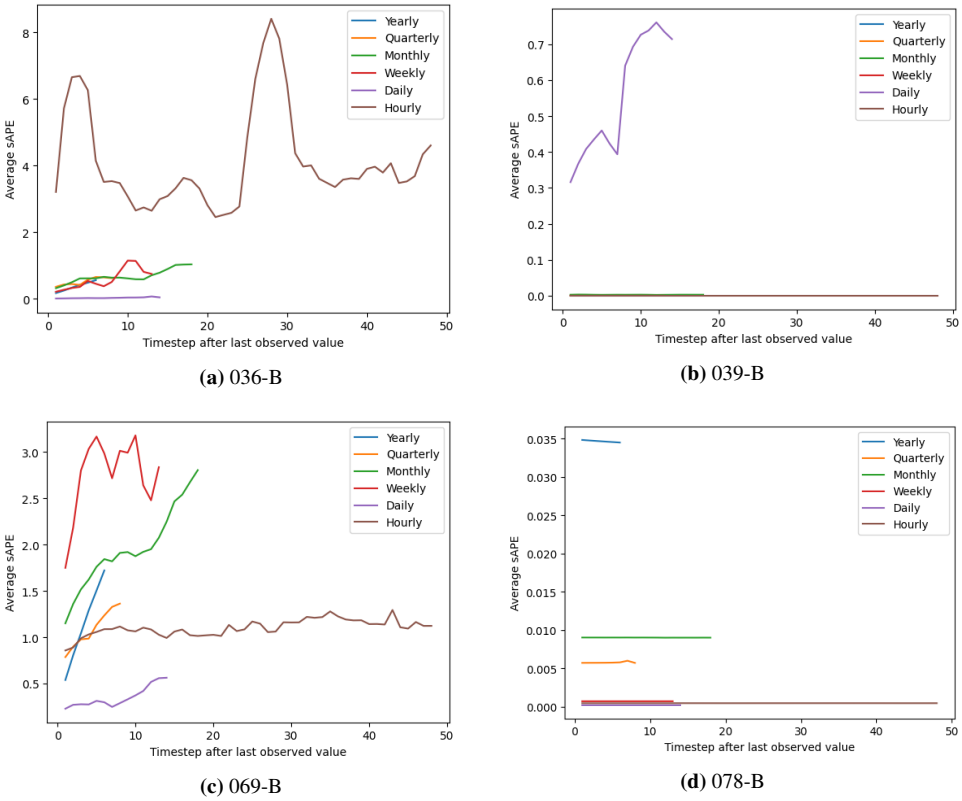
Table B.1: The table contains all 49 contributions to the M4 competition, arranged from best performing to worst performing method. The column ”Replicability category” is the category given by Makridakis et al. (2019). The category ”Unknown” could be that they did not succeed in replicating the method for various reasons, whereas the category ”Not considered” indicated that they were not mentioned in the original table of Makridakis et al. (2019) at all. The table also shows which of the methods that are publicly available at the time of writing and the time estimated by Makridakis et al. (2019) for some of the methods.

Appendix C

Results

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	-0.00044	0.00060	-0.00049	-0.00045	0.00057	-0.00146	**0.00008**
Quarterly	-0.00068	0.00004	0.00078	0.00135	-0.00041	0.00156	**0.00033**
Monthly	-0.00559	0.00124	-0.00087	0.00347	0.00609	-0.00721	**0.00195**
Weekly	-0.04589	-0.00374	-0.02935	-0.01937	0.00306	-0.04942	**-0.01267**
Daily	-0.00041	0.00134	0.00032	0.00033	-0.00066	-0.00051	**0.00034**
Hourly	NA	NA	NA	NA	NA	0.02268	**0.02268**
Total	**-0.00336**	**0.00068**	**-0.00041**	**0.00167**	**0.00258**	**0.00234**	**0.00099**

Table C.1: Difference between OWA values of the original submission and average OWA of the five reruns for method 036 on computer A.



(a) 036-B

(b) 039-B

(c) 069-B

(d) 078-B

Figure C.1: Average sAPE between the original submission and five reruns for each step in the horizon. The x-axis shows the timesteps in the forecasting horizon and the y-axis shows the sAPE.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	-0.00044	0.00060	-0.00049	-0.00045	0.00057	-0.00146	**0.00008**
Quarterly	-0.00068	0.00004	0.00078	0.00135	-0.00041	0.00156	**0.00033**
Monthly	-0.00559	0.00124	-0.00087	0.00347	0.00609	-0.00721	**0.00195**
Weekly	-0.04589	-0.00374	-0.02935	-0.01937	0.00306	-0.04942	**-0.01267**
Daily	-0.00041	0.00134	0.00032	0.00033	-0.00066	-0.00051	**0.00034**
Hourly	NA	NA	NA	NA	NA	0.02268	**0.02268**
Total	**-0.00336**	**0.00068**	**-0.00041**	**0.00167**	**0.00258**	**0.00234**	**0.00099**

Table C.2: Difference between OWA values of the original submission and average OWA of the five reruns for method 036 on computer B.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	0.00000	0.00000	0.00000	0.00001	0.00000	0.00000	**0.00000**
Quarterly	0.00000	0.00000	-0.00005	-0.00001	-0.00001	0.00000	**-0.00001**
Monthly	0.00002	0.00000	-0.00001	0.00001	-0.00001	-0.00004	**0.00000**
Weekly	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
Daily	-0.01992	0.01520	-0.01905	0.01246	-0.04091	-0.00177	**-0.00757**
Hourly	NA	NA	NA	NA	NA	0.00000	**0.00000**
Total	**-0.00011**	**0.00042**	**-0.00052**	**-0.00002**	**-0.00189**	**-0.00059**	**-0.00051**

Table C.3: Difference between OWA values of the original submission and average OWA of the five reruns for method 039 on computer A.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	0.00000	0.00000	0.00000	0.00001	0.00000	0.00000	**0.00000**
Quarterly	0.00000	0.00000	-0.00005	-0.00001	-0.00001	0.00000	**-0.00001**
Monthly	0.00002	0.00000	-0.00001	0.00001	-0.00001	-0.00004	**0.00000**
Weekly	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
Daily	-0.01992	0.01520	-0.01905	0.01246	-0.04091	-0.00177	**-0.00757**
Hourly	NA	NA	NA	NA	NA	0.00000	**0.00000**
Total	**-0.00011**	**0.00042**	**-0.00052**	**-0.00002**	**-0.00189**	**-0.00059**	**-0.00051**

Table C.4: Difference between OWA values of the original submission and average OWA of the five reruns for method 039 on computer B.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	-0.00246	-0.00038	0.00009	0.00117	0.00279	-0.00474	**0.00048**
Quarterly	0.00275	0.00037	-0.00274	0.00392	-0.00104	-0.01001	**0.00013**
Monthly	-0.00660	0.00149	-0.00062	-0.00025	0.00557	-0.01034	**0.00098**
Weekly	-0.00673	0.00371	-0.01161	0.00175	-0.00544	-0.00683	**-0.00080**
Daily	-0.00014	0.00134	0.00121	0.00066	-0.00099	0.00140	**0.00069**
Hourly	NA	NA	NA	NA	NA	0.03358	**0.03358**
Total	**-0.00314**	**0.00039**	**-0.00049**	**0.00136**	**0.00333**	**0.00138**	**0.00099**

Table C.5: Difference between OWA values of the original submission and average OWA of the five reruns for method 069 on computer A.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	0.00038	-0.00012	0.00003	-0.00022	0.00039	0.00076	**0.00009**
Quarterly	0.00077	0.00034	-0.00056	-0.00007	0.00012	-0.00089	**0.00003**
Monthly	-0.00025	0.00328	0.00201	0.00087	0.00092	0.00222	**0.00164**
Weekly	-0.00424	-0.03376	-0.00398	0.00524	0.00150	0.02999	**-0.00658**
Daily	-0.00093	0.00026	0.00192	-0.00009	0.00017	0.00473	**0.00107**
Hourly	NA	NA	NA	NA	NA	-0.00032	**-0.00032**
Total	**0.00018**	**0.00120**	**0.00089**	**0.00037**	**0.00051**	**0.00071**	**0.00073**

Table C.6: Difference between OWA values of the original submission and average OWA of the five reruns for method 078 on computer A.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	0.00284	0.00480	0.00130	0.00685	-0.00053	0.01783	**0.00366**
Quarterly	-0.00212	-0.00229	-0.00241	-0.00347	-0.00388	0.00291	**-0.00282**
Monthly	-0.02077	-0.00601	-0.00950	-0.02085	-0.03389	-0.00508	**-0.01803**
Weekly	0.00269	-0.07647	0.00463	0.06478	0.02073	-0.01570	**-0.01691**
Daily	-0.02884	-0.04378	-0.00635	-0.01487	0.05624	-0.01555	**-0.00575**
Hourly	NA	NA	NA	NA	NA	-0.00500	**-0.00500**
Total	**-0.01023**	**-0.00374**	**-0.00424**	**-0.00780**	**-0.01118**	**0.00548**	**-0.00660**

Table C.7: Difference between OWA values of the original submission and average OWA of the five reruns for method 118 on computer A.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	0.00312	0.00511	0.00099	0.00630	0.00010	0.01560	**0.00368**
Quarterly	-0.00260	-0.00243	-0.00307	-0.00298	-0.00326	-0.00320	**-0.00290**
Monthly	-0.01990	-0.00619	-0.00893	-0.02126	-0.03810	-0.00121	**-0.01895**
Weekly	0.00378	-0.07565	0.00054	0.06719	0.01610	-0.01205	**-0.01737**
Daily	-0.02674	-0.04318	-0.00589	-0.01644	0.05475	-0.01529	**-0.00586**
Hourly	NA	NA	NA	NA	NA	-0.00488	**-0.00488**
Total	**-0.00986**	**-0.00357**	**-0.00425**	**-0.00812**	**-0.01242**	**0.00393**	**-0.00698**

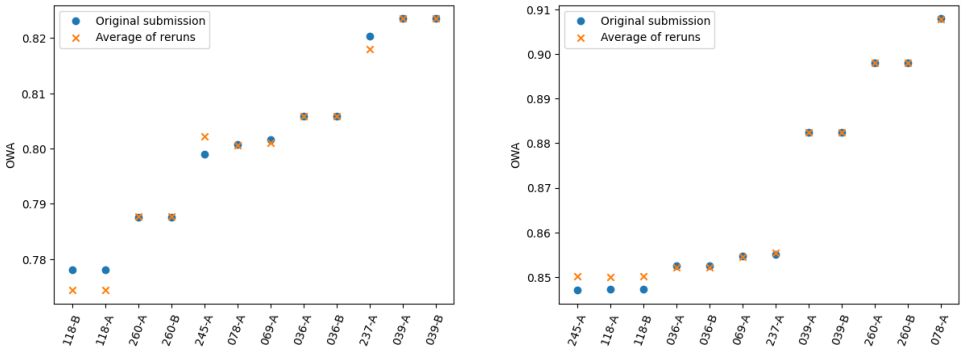
Table C.8: Difference between OWA values of the original submission and average OWA of the five reruns for method 118 on computer B.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	0.00401	0.00155	0.00214	0.00292	0.00368	-0.00046	**0.00244**
Quarterly	-0.00052	-0.00007	-0.00042	0.00071	-0.00102	0.00119	**-0.00022**
Monthly	0.00013	-0.00078	-0.00022	0.00010	-0.00010	0.00018	**-0.00023**
Weekly	0.00000	0.00107	0.00000	-0.00033	-0.00435	0.00000	**-0.00096**
Daily	0.00520	0.00451	-0.00449	0.00543	0.00082	-0.00263	**0.00131**
Hourly	NA	NA	NA	NA	NA	0.00000	**0.00000**
Total	**0.00089**	**0.00066**	**0.00049**	**0.00130**	**0.00125**	**-0.00042**	**0.00087**

Table C.9: Difference between OWA values of the original submission and average OWA of the five reruns for method 237 on computer A.

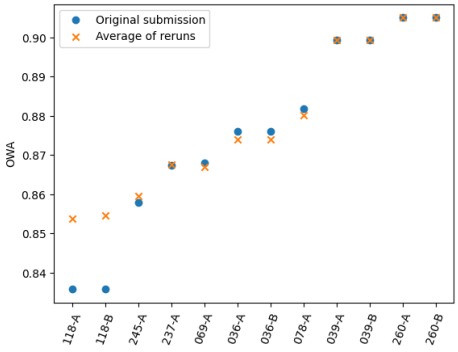
	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	-0.00685	-0.00174	-0.00291	-0.00392	-0.00353	-0.00676	**-0.00322**
Quarterly	-0.00194	-0.00470	-0.00431	-0.00188	-0.00274	-0.00271	**-0.00322**
Monthly	-0.00299	-0.00565	-0.00187	-0.00106	0.00196	-0.00413	**-0.00171**
Weekly	-0.01963	-0.00738	0.00436	-0.01595	-0.00341	-0.00058	**-0.00877**
Daily	-0.07653	0.00468	-0.00179	-0.00289	0.00094	0.00038	**0.00145**
Hourly	NA	NA	NA	NA	NA	-0.00009	**-0.00009**
Total	**-0.00420**	**-0.00267**	**-0.00259**	**-0.00209**	**-0.00105**	**-0.00381**	**-0.00226**

Table C.10: Difference between OWA values of the original submission and average OWA of the five reruns for method 245 on computer A.

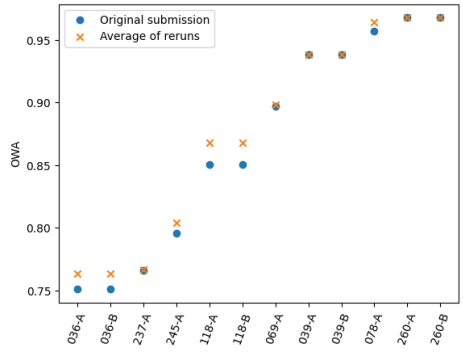


(a) Yearly

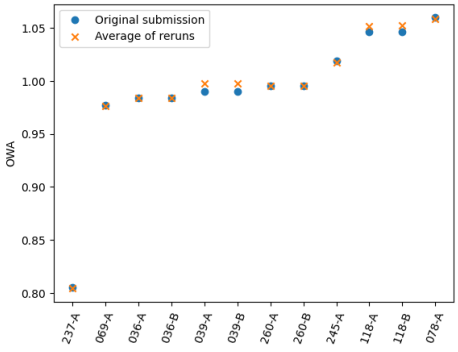
(b) Quarterly



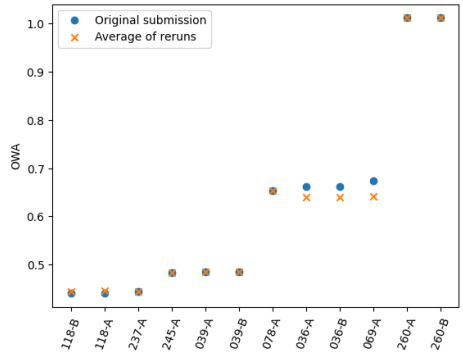
(c) Monthly



(d) Weekly

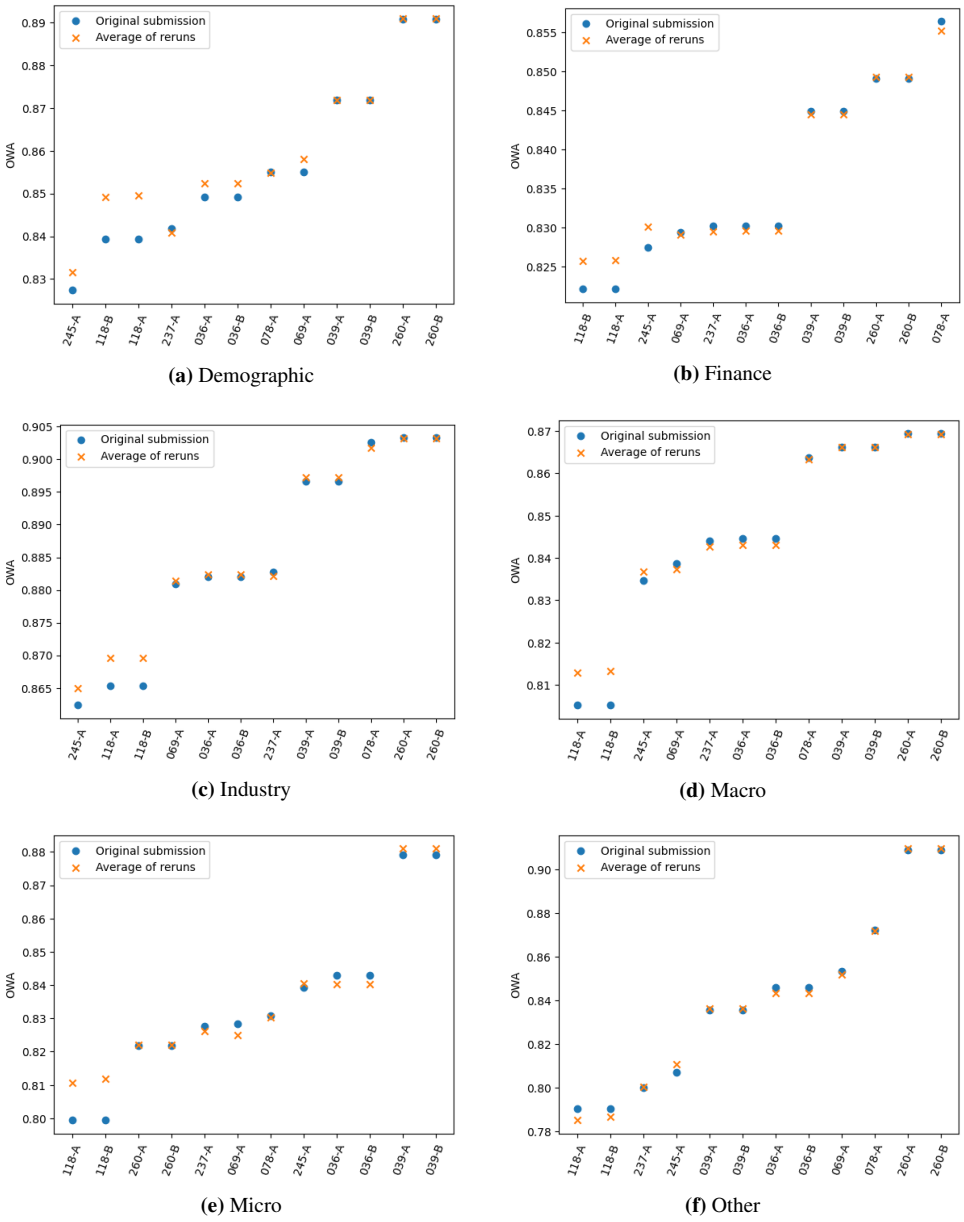


(e) Daily



(f) Hourly

Figure C.2: Average OWA for the different resolutions for the original submissions and for the reruns.



(a) Demographic

(b) Finance

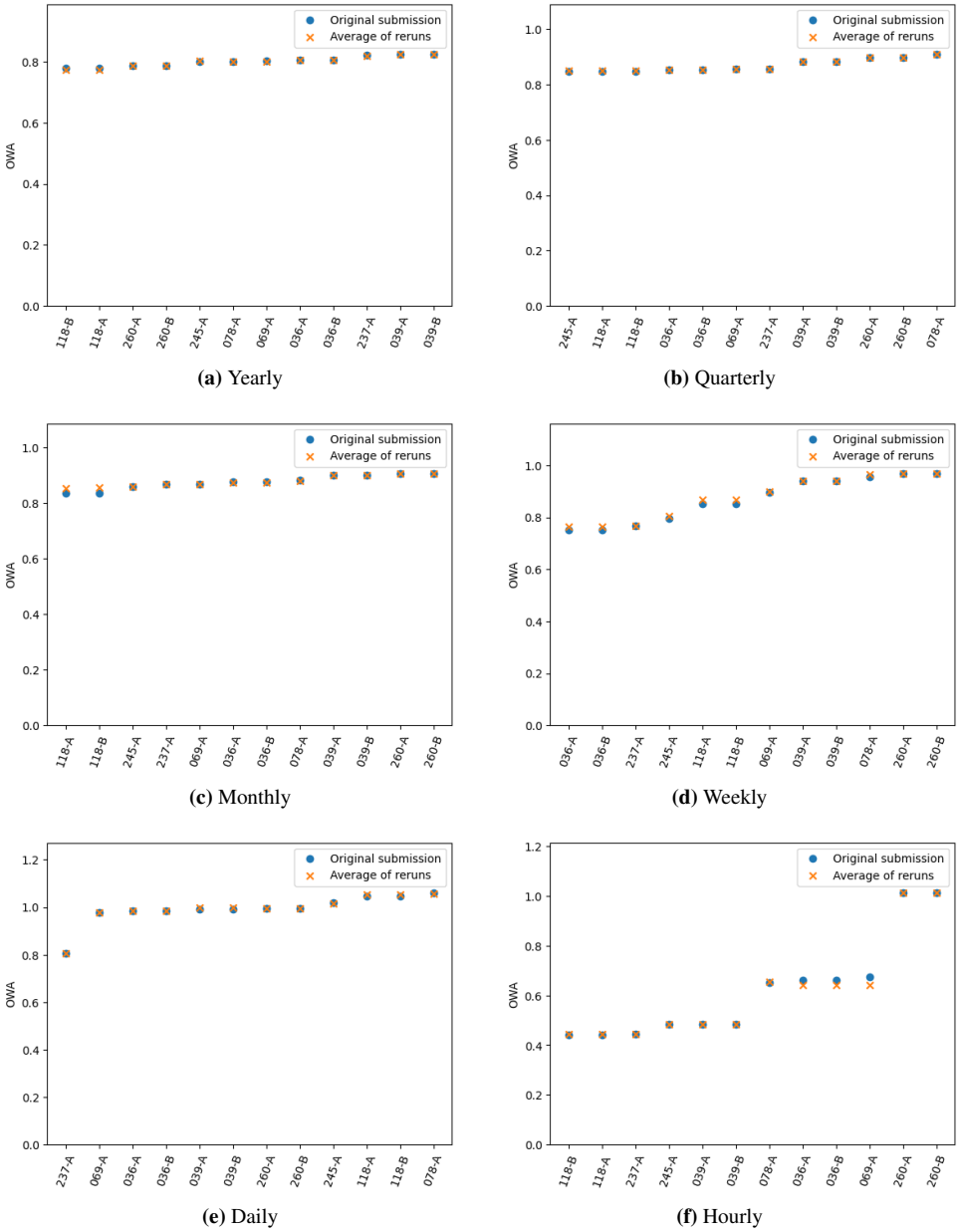
(c) Industry

(d) Macro

(e) Micro

(f) Other

Figure C.3: Average OWA for the different origins for the original submissions and the reruns.



(a) Yearly

(b) Quarterly

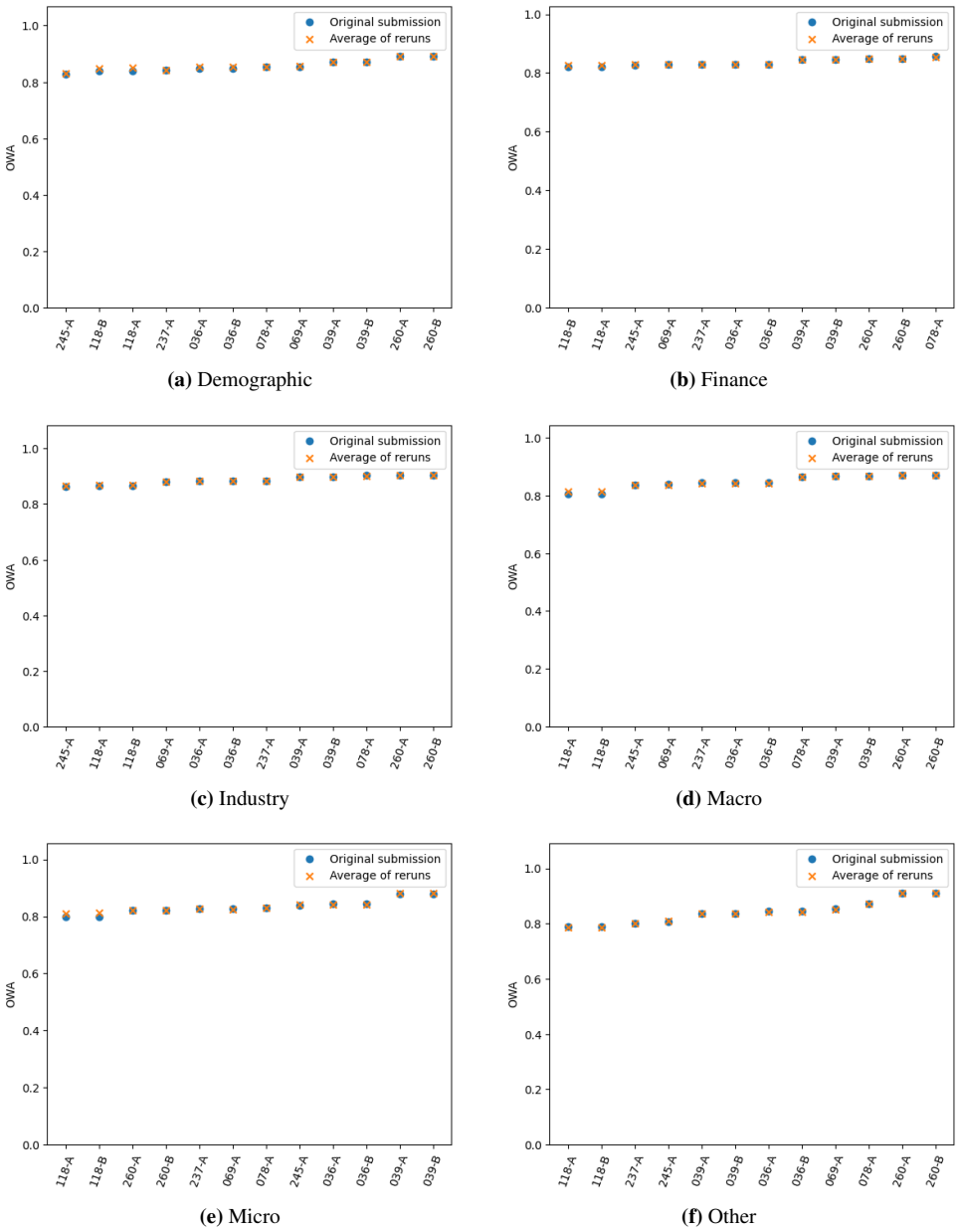
(c) Monthly

(d) Weekly

(e) Daily

(f) Hourly

Figure C.4: Average OWA for the different resolutions for the original submissions and for the reruns.



(a) Demographic

(b) Finance

(c) Industry

(d) Macro

(e) Micro

(f) Other

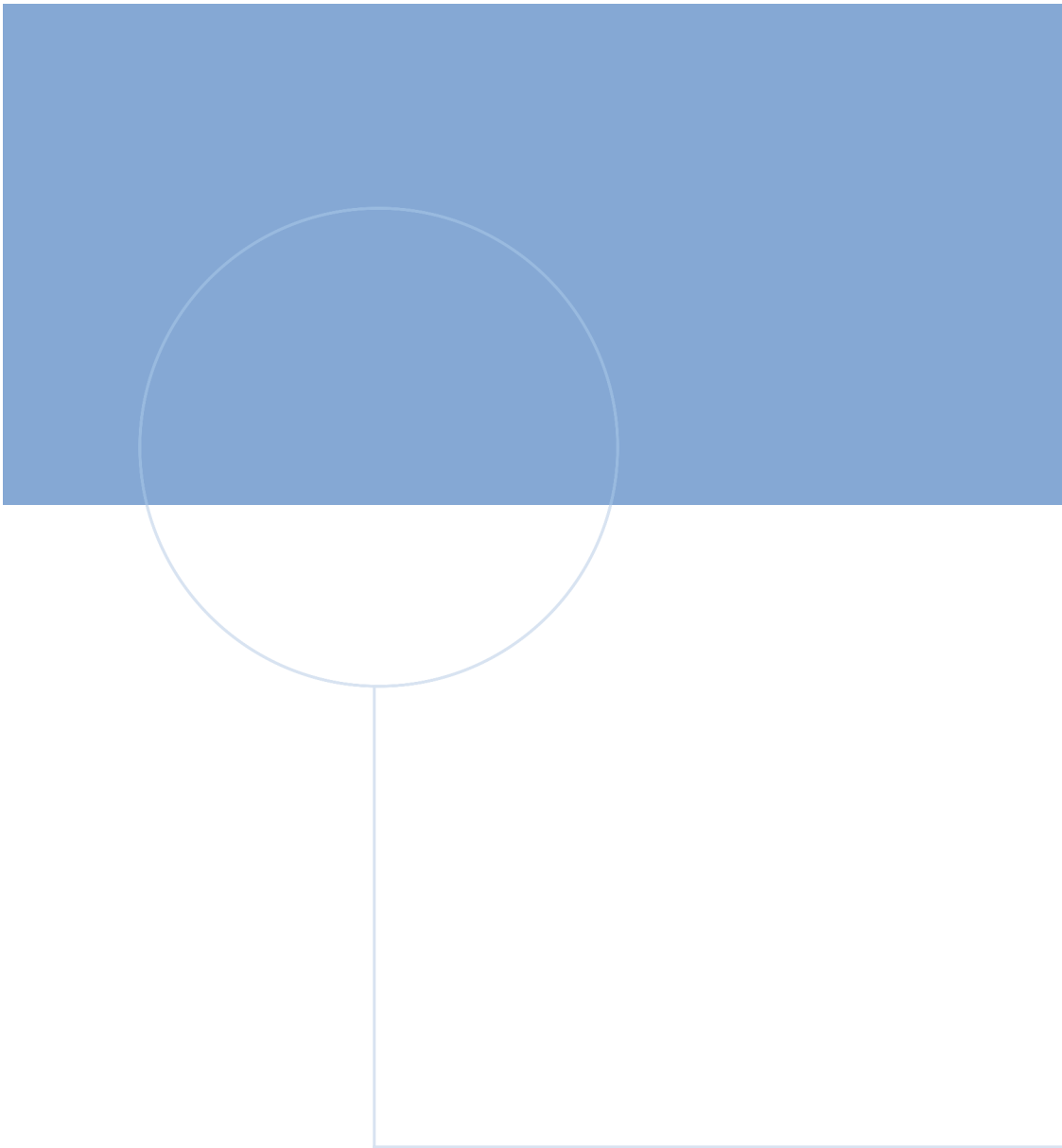
Figure C.5: Average OWA for the different origins for the original submissions and for the reruns.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	-0.00089	-0.00054	0.00051	0.00014	-0.00030	-0.00092	**-0.00020**
Quarterly	-0.00006	0.00014	-0.00024	-0.00001	0.00001	0.00000	**-0.00001**
Monthly	0.00009	-0.00006	0.00000	0.00006	-0.00006	-0.00033	**-0.00001**
Weekly	0.00000	0.00001	-0.00001	0.00000	-0.00001	0.00002	**0.00000**
Daily	0.00001	0.00000	0.00000	0.00000	-0.00001	-0.00023	**-0.00004**
Hourly	NA	NA	NA	NA	NA	0.00000	**0.00000**
Total	**-0.00016**	**-0.00018**	**0.00012**	**0.00003**	**-0.00012**	**-0.00051**	**-0.00008**

Table C.11: Difference between OWA values of the original submission and average OWA of the five reruns for method 260 on computer A.

	Demographic	Finance	Industry	Macro	Micro	Other	Total
Yearly	-0.00089	-0.00054	0.00051	0.00014	-0.00030	-0.00092	**-0.00020**
Quarterly	-0.00006	0.00014	-0.00024	-0.00001	0.00001	0.00000	**-0.00001**
Monthly	0.00009	-0.00006	0.00000	0.00006	-0.00006	-0.00033	**-0.00001**
Weekly	0.00000	0.00001	-0.00001	0.00000	-0.00001	0.00002	**0.00000**
Daily	0.00001	0.00000	0.00000	0.00000	-0.00001	-0.00023	**-0.00004**
Hourly	NA	NA	NA	NA	NA	0.00000	**0.00000**
Total	**-0.00016**	**-0.00018**	**0.00012**	**0.00003**	**-0.00012**	**-0.00051**	**-0.00008**

Table C.12: Difference between OWA values of the original submission and average OWA of the five reruns for method 260 on computer B.



NTNU

Norwegian University of
Science and Technology