Masarykova univerzita
Fakulta informatiky

Alert Prediction in Metric Data Based on Time Series Analysis

Master thesis

Bc. Pavol Loffay

Brno, spring 2016

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Pavol Loffay

Advisor: RNDr. Adam Rambousek, Ph.D.

Acknowledgement

I would like to thank my supervisor RNDr. Adam Rambousek, Ph.D., Mgr. Jiri Kremser, family and Hawkular Team. Next, my thanks goes to Ing. Daniel Němec, Ph.D. for consulting theory behind time series analysis.

Last but not least, I would like to thank to company Red Hat that provided me with great opportunity to work on this project.

Abstract

The aim of the master's thesis is to develop a module for an open source monitoring and management platform Hawkular. This module is responsible for predicting alerts based on time series and providing data for predictive charts in Hawkular user interface.

Keywords

Time Series, Hawkular, Alert Prediction, Exponential Smoothing

Contents

1 Introduction

When driving successful business on the internet, it is important to assure application's health and reliability. One can achieve that by monitoring subjected resources and setting up clever alerting system.

These features are offered by many monitoring systems, however being predictive in this area can even prevent undesirable states and most importantly gives administrators more time to react to such events. For instance it can decrease downtime of an application or ability to load balance workload in advance by horizontal scaling of targeted services.

Alerting systems are sophisticated and can be composed by many conditions. This work focuses on predicting future metric values which are then sent as input for evaluation to alerting system.

The work starts with time series theory by describing various approaches for time series modelling and forecasting. The chapter describes models which are used within the work, could be used or explains why the decision was made to use other models.

Following chapter demonstrates models from previous chapter on real time series. It visually shows predictive capabilities of targeted models and compares produced statistical quantities.

The third chapter briefly describes existing monitoring and management solutions which are to some extend Hawkular competitors. Java libraries for time series forecasting are also mentioned there.

The fourth chapter describes implementation part of the thesis. It starts with core artifacts for time series forecasting and continues to building web application and integration into Hawkular.

The last chapter is dedicated to the evaluation of the forecasting accuracy of the implemented solution. Models from `forecast` package from language R were chosen as a referential implementation.

1.1 Hawkular

The implementation part of the master's thesis is developed as a part of an open source project Hawkular[1], therefore, the application archi-

1. Available at `http://www.hawkular.org`

tecture and used technologies had to fit into the overall project design.

Hawkular is middleware monitoring and management platform developed by company Red Hat and independent community of contributors. It is a successor of very successful RHQ[2] project, also known as JBoss Operations Network (JON). By monitoring, it is meant that there are agents for diverse applications which push data to the server. These agents can also execute application specific actions.

The monolithic architecture of the RHQ project was due to its size hard to maintain and lacking robust REST API lead to fresh development of new application. Hawkular, in contrast, consists of several loosely coupled or even independent applications. These independent components are much easier to maintain and what is more important they communicate over REST API.

This architecture of microservices and chosen protocol allow simple development of agents which can be written in any programming language. In RHQ only Java agent were available.

Hawkular as a product itself is customized Wildfly[3] application server with all components deployed in it.

List of Hawkular main components:

- Console – user web interface.
- Accounts – authorization subsystem based on Keycloak[4].
- Inventory – graph based registry of all entities in Hawkular.
- Metrics – time series metrics engine based on Cassandra[5].
- Alerts – alerting subsystem based on JBoss Drools.

Some of the modules also use Java messaging system (JMS) for inter-component communication.

---

2. Available at `https://rhq-project.github.io/rhq/`.
3. An open source project of JBoss Enterprise Application Platform.
4. An open source single sing-on and identity management for RESTful web services.
5. An open source distributed database management system. Hybrid between key-value and column-oriented database.

Modules are packaged as standard Java web archives (WAR), or enterprise archives (EAR) and deployed into Wildfly. Build and package management is performed by Maven and Gulp for user interface modules.

1.2 Data Mining Goals

The goal of this thesis is to develop a module for Hawkular which will provide forecasts for any time series metrics collected by agent.

Module should learn from new incoming metric data and immediately send predicted values for future timestamps to alerting subsystem. Based on this values an alert can be triggered. This implies that forecasting engine should work on continuous flow of data. Forecast should be also available for user interface in predictive charts.

One Wildfly agent on average collects hundreds to thousands metrics, therefore module should be capable of processing high volume of data. Some of the customers monitor hundreds of servers, each with multiple agents. Therefore performance of chosen learning algorithm has to be taken in the account.

1.3 Metrics in Hawkular

In Hawkular, there are three types of metrics: gauge, counter and availability. All of them are univariate metrics of structure $\{timestamp, value\}$. Each of these types is used for collecting dedicated types of metric data. For example gauge can increase or decrease over the time, counter monotonically decreases or increases and availability represents up or down state of a resource.

2 Time Series Models

This chapter focuses on time series theory and various approaches for time series modelling. Models are put in order from simpler to more complex ones. Only some of the discussed models are selected for the implementation. Chapter also adds some theory necessary for time series analysis, for example unit root tests, seasonal decomposition and period identification of seasonal time series.

From the goals from Section 1.2 results that selected time series models should be robust enough to work on arbitrary monitored time series and at the same time to be computationally inexpensive when it comes to learning and forecasting process.

Firstly, it is important to define time series; it is a sequence of observations $s_t \in \mathbb{R}$ put in order of time. This thesis focuses only on univariate equidistant discrete time series.

Time series analysis contains many segments, this work aims on forecasting only. It is defined as a process of making prediction of the future based on the past. In other words, forecasting is possible because future depends on the past or analogously because there is a relationship between the future and the past. However, this relation is not deterministic and can be hardly written in an analytical form [10].

Generally there are two forecasting types: qualitative and quantitative. Qualitative methods are mainly based on the opinion of the subject and are used when past data are not available, hence not suitable for this project. If there are past data available, quantitative forecasting methods are more suitable.

2.1 Simple Quantitative Models

Following methods are the simplest forecasting quantitative models. They can be used for any time series without any prerequisites or further analysis.

- Average method – forecasts are equal to the value of the mean of historical data.

$$\hat{y}_{T+h|T} = \overline{y} = (y_1 + \cdots + y_T)/T \tag{2.1}$$

- Naïve method – forecasts are equal to the last observed value.

$$\hat{y}_{T+h|T} = y_T \tag{2.2}$$

- Drift method – variation of naïve method which allow the forecasts to increase or decrease over time.

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T} y_t - y_{t-1} = y_T + h(\frac{y_T - y_1}{T-1}) \tag{2.3}$$

There are also seasonal variants of these models which hold one model for each season. These methods in general produces high forecasting error but are very easy to implement.

2.2 Linear Regression

Linear regression is a classical statistical analysis technique. It is mostly used to determine whether there is linear relationship between dependent and eventually more independent variables [15]. It is also used for predictions mainly in econometric field.

Simple linear regression is defined as:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{2.4}$$

Parameters β_0 and β_1 are calculated by minimizing the sum of squared errors:

$$SSE = \sum_{i=1}^{N} \epsilon_i^2 = \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i)^2 \tag{2.5}$$

Once the parameters are estimated, predictions for any time in the future can be calculated. If modelled time series is not stationary and trend changes over time parameters should be periodically reestimated to achieve better forecasting accuracy. This means the regression model does not adapt if the underlying time series changes.

2.3 Simple Exponential Smoothing

The concept behind simple exponential smoothing is to attach larger weights to the most recent observations than to the observations from distant past. Forecasts are calculated using weighted averages where the weights decrease exponentially as observations come from further in the past [12]. In other words, smaller weights are associated with older observations. Equation for simple exponential smoothing is listed in 2.6.

$$\begin{aligned} \hat{y}_{T+1|T} &= l_t \\ l_t &= \alpha y_t + (1-\alpha) l_{t-1} \end{aligned} \tag{2.6}$$

For smoothing parameter α holds $0 \leq \alpha \leq 1$. Note, if $\alpha = 1$ then $\hat{y}_{T+1|T} = y_T$ so forecasts are equal to the naïve method. If the parameter α is smaller more weight is given to the observations from distance in the past.

Simple exponential smoothing has a flat forecast function, that means all forecasts are the same. Therefore this method is useful if a series does not contain any trend or one is interested only in one step ahead prediction. Multi step ahead predictions for time series with trend usually produce high error rate.

This model is generally used for noise separation or trend estimation.

2.4 Holt's Liner Trend Model

Simple exponential smoothing can be extended to allow forecasting of data with a trend. This was done by Charles C. Holt in 1957. This method is slightly more complicated than the original one without trend. In order to add trend component another equation has to be added 2.7.

$$\begin{aligned} \hat{y}_{t+h|t} &= l_t + h b_t \\ l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta (l_t - l_{t-1}) + (1-\beta) b_{t-1} \end{aligned} \tag{2.7}$$

Parameter b_t denotes a slope of the series and the parameter l_t level. There is also a new smoothing parameter for the slope – β. Its rage is equal to α, so $\alpha, \beta \in [0,1]$.

2.5 Holt-Winters Seasonal Model

This model is an extension of Holt's linear trend method with added seasonality. It is also called triple exponential smoothing. There are three equations in this model 2.8. One for level, second for trend and third for seasonality. Each pattern uses smoothing constant $\alpha, \beta, \gamma \in [0,1]$.

$$\begin{aligned} \hat{y}_{t+h|t} &= l_t + hb_t + s_{t+h_m-m} \\ l_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \\ s_t &= \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m} \end{aligned} \tag{2.8}$$

Where $h_m = [(h-1) \mod m] + 1$, which ensures that the estimates of the seasonal indices came from the correct season. This model can be used only if the period of time series is know beforehand. In Hawkular the period of the time series is unknown, therefore period identification should be also implemented.

Each exponential smoothing model contains parameters which should be precisely estimated (α, β, γ), but it is also possible to use default values from literature. However, with default values models produce higher forecasting error than with estimated parameters. Estimation process defines objective function which value can be mean squared error, mean absolute error or likelihood. Then this objective function is passed to non-linear optimization algorithm.

Time complexity of exponential smoothing models for learning N observations is $O(N)$. With enabled optimization the complexity depends on used optimization algorithm.

Exponential smoothing models have advantage over other more complicated models that if a system needs to achieve high performance than default parameters can be used. This would not be possible with ARIMA models for instance.

2.6 Box – Jenkins Methodology (ARIMA)

Models from Box – Jenkins methodology are the most widely used in time series analysis for econometric data. This methodology is based on analysis of autocorrelation (ACF) and partial autocorrelation (PACF) functions.

The most generic model is ARIMA(p, d, q). It combines autoregressive, integrated and moving average parts together. An autoregressive model (AR) consist of sum of weighed lagged observations 2.9. The order of this model is defined by p and can be determined from PACF function [15].

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t \qquad (2.9)$$
$$\epsilon_t \overset{iid}{\sim} N(0, \sigma^2)$$

A moving average model (MA) is a sum of weighted errors of order q. The order of this part can be determined from ACF function [15]. Moving averages model should not be confused with simple moving average from 2.3 which is used for trend estimation. In moving average model MA(q) the current value is a regression against white noise of prior values of the series [16]. A random noise from each point is assumed to come from the same distribution which typically is a normal distribution. Model of the order q is listed in 2.10.

$$y_t = \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_p y_{t-q} + \epsilon_t \qquad (2.10)$$
$$\epsilon_t \overset{iid}{\sim} N(0, \sigma^2)$$

The last part of the model is used when a time series is non stationary. There are several ways how to make a particular time series stationary. Box – Jenkins methodology uses differencing – integration part I(d). The order of differentiated original series is denoted by d letter. Usually first order differences are enough to make a time series stationary.

It is important to mention that models AR(p) and MA(q) are invertible. Therefore any stationary AR(p) model can be written as MA(∞) and with some assumptions vice versa [7]. ARIMA model is often

written with backshift operator $By_t = y_{t-1}$. With this operator ARIMA(p, d, q) is listed in 2.11. On the left side of the equation is AR(P) process and on the right MA(q).

$$(1 - \phi_1 B - \cdots - \phi_p B^p)(1 - B)^d y_t = \\ c + (1 + \theta_1 B + \cdots + \theta_q B^q)\epsilon_t \tag{2.11}$$

The parameters of the model, including AR and MA part can by estimated by non-linear optimization algorithm with likelihood as objective function [7]. However this function is much more complicated than for exponential smoothing models. For successful estimation a certain number of historical points needs to be available. Reference [15] advises minimal training size of at least fifty observations.

After this theoretical part it is clear that ARIMA models are more complicated than family of moving averages. To this also refers [11] and adds that ARIMA models are less robust than exponential smoothing models.

2.7 Artificial Neural Networks

Recently a large number of successful applications using neural networks for time series modeling show that they can produce valuable results [17]. There are several non trivial issues with determining the appropriate architecture of the network. This has to be taken into account because it can dramatically effect learning performance and forecasting accuracy [6]. Besides the problems with selecting right architecture learning process of artificial neural network is much more computationally expensive than selecting appropriate ARIMA or exponential smoothing model [8].

Because Hawkular forecasting engine should be capable of predicting thousands of metrics at the same time, models based on neural networks would have too high computational requirements, therefore, they are not suitable for our environment.

2.8 Time Series Decomposition

In modelling time series it is sometimes necessary to decompose series to trend, seasonal and random component [10]. It is also used for initialization seasonal indices in triple exponential smoothing.

- **Trend** T_t – exists if there is long term increase or decrease over time. Can be linear or nonlinear (e.g. exponential growth).
- **Seasonal** S_t – exists when a series is influenced by seasonal factors. Seasonality is always of fixed and known period.
- **Cyclic** C_t – exists if there are long term wave-like patterns. Waves are not of a fixed period.
- **Irregular** E_t – unpredictable random value referred as white noise.

Decomposition can be written in many forms. Two of them are additive 2.12 and multiplicative 2.13. Which one to use depends on the underlying time series model.

$$y_t = T_t + S_t + C_t + E_t \tag{2.12}$$

$$y_t = T_t \times S_t \times C_t \times E_t \tag{2.13}$$

An algorithm for additive decomposition consists of following steps:

1. Compute a trend component $\hat{T}_t$ using moving average model. If a period is even use $2xMA(period)$. If period is an odd use $MA(period)$. $2xMA$ for even period is used because it has to be symmetric.
2. Calculate detrended series $y_t - \hat{T}_t$.
3. Estimate seasonal indices $\hat{S}_t$ for each period by averaging values of given period. For example, the seasonal index for Monday is the average for all detrended Monday values in the data. Then the mean of seasonal indices is subtracted from each period.
4. Random component is calculated by subtracting trend and seasonal component from original time series $\hat{E}_t = y_t - \hat{T}_t - \hat{S}_t$.

2.9 Augmented Dickey – Fuller Test

Time series statistical tests are often used for testing if there is particular characteristics present in time series. Unit root tests are used whether a time series is stationary. In this work Augmented Dickey – Fuller (ADF) test was chosen for unit root testing.

Its null hypothesis is H_0*: time series contains a unit root* – it is not stationary. Outcome of this test is a negative ADF statistics. The more negative it is the stronger the rejection of the hypothesis. The full form of ADF test is listen in 2.14.

$$\Delta y_t = \alpha + \beta t + \gamma \Delta y_{t-1} + \cdots + \delta p - 1 \Delta y_{t-p+1} + \epsilon_t \tag{2.14}$$

$$ADF = \frac{\hat{\gamma}}{SE(\hat{\gamma})} \tag{2.15}$$

There are multiple variants of ADF test. Some of them leave out parts of equation 2.14. The most important ones and widely used are:

- *nc* – for regression with no constant nor time trend (βt)
- *c* – for regression with an intercept but no time trend (βt)
- *ct* – for regression with an intercept and time trend

Each of them is good when testing particular type of stationarity. For example for testing if there is trend present in the time series *c* version is the best choice.

The implementation of this test fits multiple linear regression model from equation 2.14 and calculates ADF statistics listed in 2.15. SE denotes standard error of estimated $\hat{\gamma}$.

2.10 Seasonality Detection

Forecasting engine in Hawkular system does not have any inside information about period of a time series being modelled, therefore, automatic period identification has to be implemented. In practice it is a difficult task and result often differs from correct period, especially if there is significant noise present in the series [14].

There are several approaches how to implement automatic period identification. The most used ones are based on autocorrelation function (ACF) or spectral density [2]. This work applies ACF method.

In the following Chart 2.1 ACF function of sine function is shown. The period of this function is seven. There are patterns repeated every seven observations and it decreases to zero.

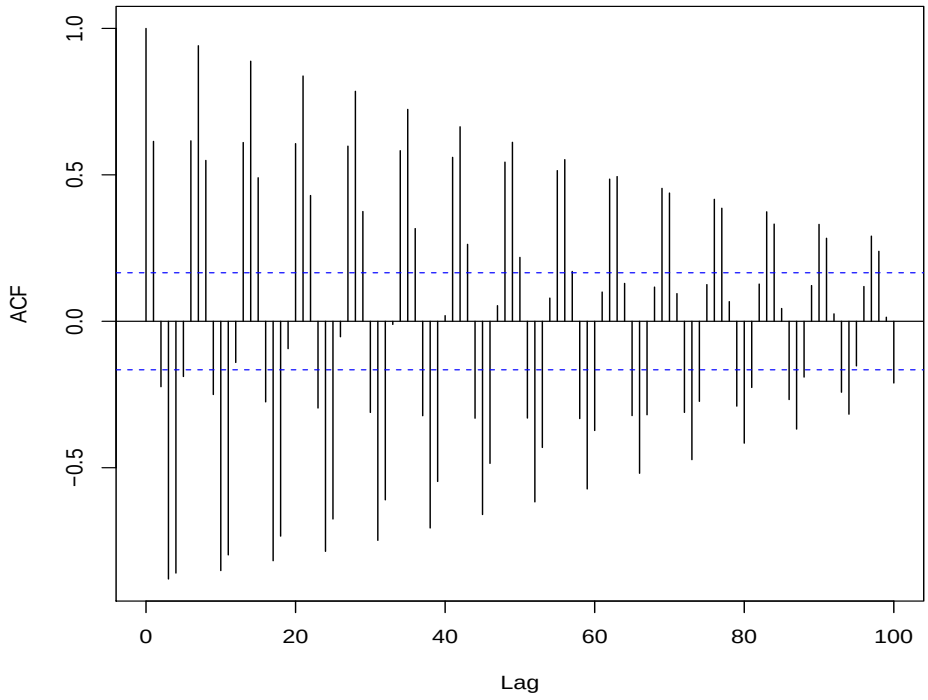

Figure 2.1: Autocorrelation function of sine function.

The algorithm for automatic period identification is demonstrated in Algorithm 1. It looks for periodically repeated significant values of ACF function. At some rate these values have to be decreased to zero. In the infinity ACF function converges to zero.

The algorithm as input takes only time series. If there is a unit

root present, then differencing is applied. Calculation of autocorrelation function of input time series follows and finding the index of its highest value. Then it checks if there are significant values of ACF present at following $n * period$ indices. There have to be present at lease two consecutive values of ACF, so n takes values from $1, 2, 3 \ldots$

Algorithm 1 Find period of time series

```
1:  function findFrequency(int[]ts)
2:      if unitRootPresent(ts) then                    ▷ e.g. ADF test
3:          ts ← diff(ts)                              ▷ first order differences
4:      end if
5:      acf ← acf(ts)
6:                                  ▷ returns index of the highest value
7:      period ← findHighest(ts, period)
8:      while period * 2 < ts.length do
9:          if checkPeriodExists(x, ts) then
10:             return period
11:         end if
12:         period ← findHighest(ts, period)
13:     end while
14:     return 1
15: end function
```

3 Analytical Forecasting Process

In the previous chapter several time series models were described. However, in Hawkular only few of them were selected and implemented.

This chapter demonstrates targeted models on real time series. There is also conducted a statistical comparison of the models. This chapter also contains theory how different models should be compared. At the end there is a discussion which models were selected for usage in Hawkular.

3.1 Evaluating Forecasting Accuracy

In order to evaluate a forecasts produced by particular model it is important to calculate the errors of the forecasts. There are several statistics for evaluating forecasting accuracy. The most used ones are mean squared error (MSE) 3.1 and mean absolute error (MAE) 3.2 [9]. The difference between them is that MSE emphasizes the extremes while MAE is more robust to outliers [11].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{3.1}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{3.2}$$

3.2 Measuring Model Quality

When comparing multiple models, statistics like MSE or MAE are not the best objective functions. Comparison based on this statistics can select complicated model with lots of parameters, which may overfit training data and most importantly it selects less robust model [15]. Therefore for comparison multiple models another factors have to be added to the objective function. These factors are number of parameters of the model.

The most used criteria for model sections are: Akaike information

criterion (AIC) and Bayesian information criterion (BIC). Model with lower information criterion is preferred.

$$AIC = 2k - \ln(L) \tag{3.3}$$
$$BIC = k\ln(n) - 2\ln(L)$$

Equations for AIC and BIC are listed in 3.3. Number of parameters of the model is represented by k. Number of observations is denoted by n and L stands for maximized value of the likelihood function of the model. In case for exponential smoothing it is minimized sum of squared error of one step ahead prediction of training data set.

From the equations it can be seen that BIC penalizes models with more parameters. There is also AIC criterion with correction form 3.4. Corrected version of AIC more penalizes longer models.

$$AICc = AIC + \frac{2k(k+1)}{n-k-1} \tag{3.4}$$

3.3 Models Demonstration

An analytical process of modelling time series starts with plotting the data and fitting various models chosen by forecaster's previous experience. In second step forecaster compares statistics of those models and chooses the one that describes data the best.

In this section real time series is modelled using time series models discussed in Chapter 2. Chosen time series is *austourists* from R package `fpp`. This series is a measurement of quarterly visitor nights spent by international tourists in Australia. It was chosen on purpose because it contains trend and seasonality.

On the first Chart 3.1 there is depicted naïve, average and drift model. Average model is just an overall average of the whole time series. It does not change over time. However there could be an online version of this algorithm for which an average would be calculated for each new value.

Naïve and drift model copy values of time series with lag of one observation. These two models differs in forecasts where drift model is capable of forecasting trends.

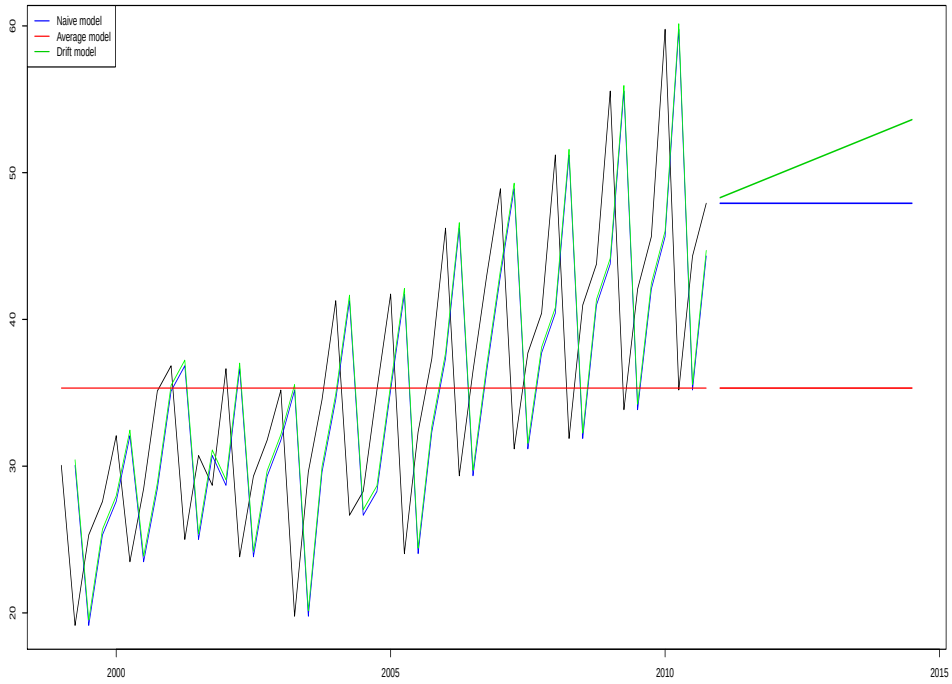

Figure 3.1: Simple models on *austourists*.

Exponential smoothing models are depicted in Chart 3.2. For this particular series it is obvious that seasonal model is the best describing modelled time series.

In the beginning it can be seen how is the seasonal model changing and learning the seasonal pattern from data. Learning and adaptivity to new trends depends on smoothing parameters of the models. Higher values of parameters allows the model quicker adapt to changes. Models with lower smoothing parameters are more robust to the changes.

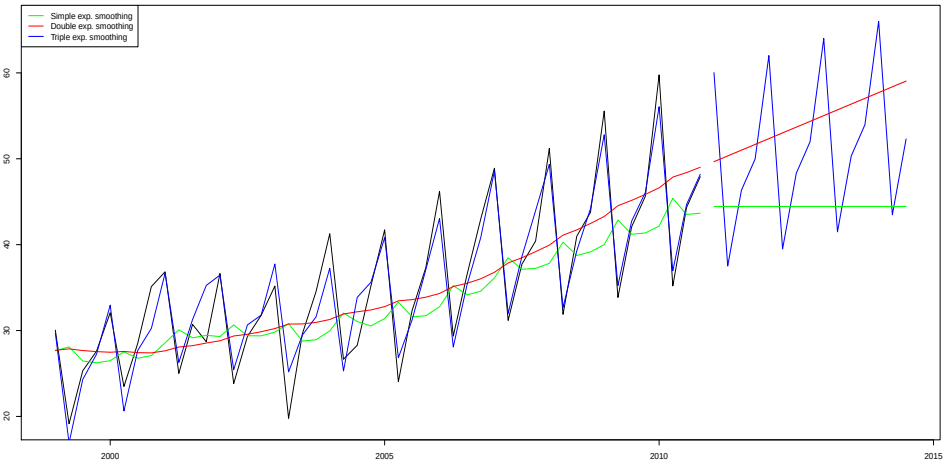

Figure 3.2: Exponential smoothing models on *austourists*.

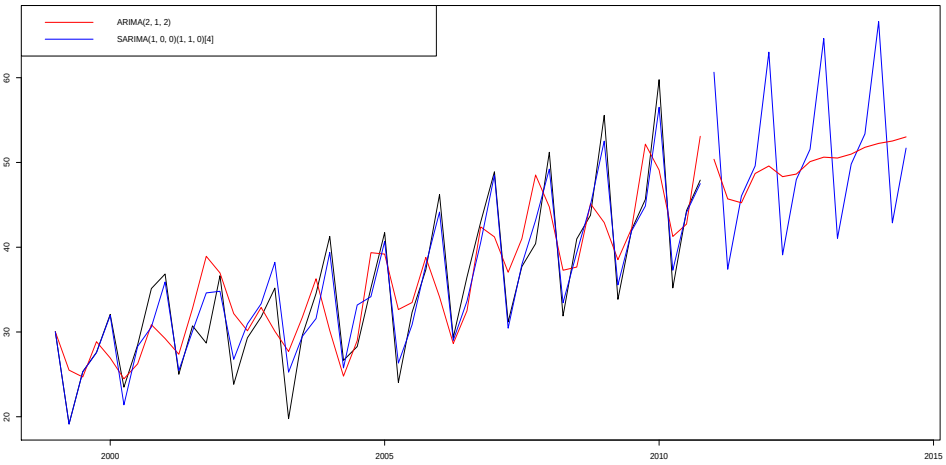

Figure 3.3: Arima models on *austourists*.

ARIMA models are shown in Chart 3.3. Non-seasonal model is already capable of capturing some seasonal patterns but it is more random whereas seasonal patterns should be of fixed period. Therefore seasonal ARIMA with dedicated model for each period is more appropriate.

Model	**MSE**	**MAE**	**AIC**
Naïve	104.74	8.64	
Drift	104.60	8.47	
Average	79.91	7.14	
Simple exp. smoothing	53.54	5.91	380.88
Double exp. smoothing	44.41	5.28	375.90
Triple exp. smoothing	5.34	1.71	282.73
Linear regression	41.85	5.10	321.46
ARIMA(2, 1, 2)	30.88	4.35	310.6
SARIMA(1, 0, 0)(1, 1, 0)[4]	4.79	1.63	206.95

Table 3.1: Statistics of models.

Statistics produced by each model are listed in Table 3.1. Models with the lowest one step ahead prediction error are seasonal ARIMA and triple exponential smoothing which is correct for this highly seasonal time series.

As it was discussed in Section 3.2 AIC criterion should be used for models comparison. From Table 3.1 it can be seen that AIC and MSE are related, however MSE of double exponential smoothing is 8.3 times higher than MSE of triple exponential smoothing, whereas AIC of double exponential smoothing is only 1.3 times higher. From this example it is obvious how AIC discriminates models with more parameters.

4 Existing Solutions

Before going directly to the design and implementation let's briefly look at existing monitoring solutions and libraries for time series modelling.

In the next section not only direct competitors of Hawkular are described but also generally available software of this kind. Because Hawkular is still only an upstream project and not deliverable product for the customers, thereby Jboss Operations Network (JON) is considered as its alternative.

However, this work does not focus on precise comparison of monitoring applications. It is only mentioned for wider overview of this domain.

4.1 Monitoring and Management Applications

There are many monitoring solutions available out there. Some of them are offered as service (SaaS) and other need to be deployed in customer's environment.

The most important features for each application are highlighted altogether with information how it differs from its competitors and if application offers predictive capabilities.

4.1.1 New Relic

New Relic was the first monitoring solution offered as SaaS [4]. It focuses on application performance rather than infrastructure view.

Monitoring of Java applications is done by agents. Because agent manipulates with bytecode it is able to identify business transactions and monitor for instance utilization of REST endpoints.

Agents also automatically discover databases connected to the application being monitored. From alert prediction perspective New Relic does not offer any alert prediction or predictive charts.

4.1.2 Dynatrace Ruxit

Ruxit is another popular SaaS solution for monitoring middleware applications. Monitoring is also done by agents which instrument bytecode so the application is also able to monitor business transactions. Ruxit differs from others in offering root cause analysis rather than firing multiple alerts [5]. It directly shows causing problems with visual representation. Ruxit does not offer any predictive alerting or charts.

4.1.3 Open Source Projects

As Hawkular is an open source project it is important to mention its direct competitors from this area. There are two major open source projects. The first one is Nagios and second Zabbix. Nagios was started in 1991 and Zabbix in 2001 therefore both are more than fifteen years old.

In comparison to Hawkular, Ruxit or New Relic the user interface of these two projects feels much older. At the time of writing this thesis last contribution to Nagios was from 17th September 2015 and Nagios project was active until the last moment. Both of these projects do not offer alert prediction.

4.1.4 HP OpenView and IBM Tivoli

The biggest monitoring solutions offered by these two software giants are HP OpenView and IBM Tivoli. Both are composed of many independent products. Solution from HP does not offer any predictive capabilities.

Operations Analytics, one of IBM Tivoli products can automatically detect correlations between metrics which can lead to detection of application issues, for example an anomaly detection [1].

Another part of Tivoli solution – Netcool Network Management is capable of predicting events. There is also another predictive analysis in Tivoli Monitoring module[1]. In the online document there it is

1. Available at `https://www.ibm.com/developerworks/community/wikis/home?lang=en#/wiki/Tivoli+Monitoring/page/Details+for+ITPA+Predictive+Analytics+and+Non+Linear+Trending`.

mentioned that predictive engine does not work on real time streams of data. Tivoli is definitely a big solution and in order to get real predictive capabilities it would require to contact IBM support.

4.2 Libraries For Time Series Forecasting

In Java there is only one publicly available library for time series modelling. The library is called OpenForecast[2]. It contains models like naïve, linear regression and exponential smoothing models. Whereas the library is open source, code was reviewed and tested. Some assumptions about the implementation were made. The parameters of some models are not estimated correctly. For example in linear regression slope and intercept are not estimated by minimizing sum of squared errors.

The library provides nice feature called automatic forecaster which selects the best model for given time series, however the code produces null pointer exception. The bug was identified but the library is not maintained anymore.

The performance of the library was also tested. Table 4.1 contains execution times of optimizers for simple, double and triple exponential smoothing models. The tested time series is generated sine function of length 200 observations. The result is compared against execution times of R's function `ets` from `forecast` package.

Model	**OpenForecast**	**R `forecast`**
Simple ex.	4.91 sec.	0.003 sec.
Double ex.	9.31 sec.	0.006 sec.
Triple ex.	6.45 sec.	0.209 sec.

Table 4.1: Execution time of parameters estimation for exponential smoothing models.

From the table it is obvious that R's implementation is much faster. For example optimization of thousand double exponential smooth-

2. Available at `http://www.stevengould.org/software/openforecast/index.shtml`.

ing models would take about two and half hours. This results are not acceptable for Hawkular in real time environment.

Another possibility is to directly use R implementation. There are various libraries[3] which allow executing R code from Java, however it requires installed R system in the target environment.

Other possible solution is to use Rengin[4] – JVM interpreter for R. However `forecast` package depends on other package which invocates native C++ code for optimized computations. Therefore `forecast` package can not be used with Rengin.

3. The most widely used: RCaller and JRI.
4. Available at `http://www.renjin.org/`

5 Design and Implementation

The module for an alert prediction is named Hawkular Data Mining. Source code is versioned in Git and hosted on Github[1] under license Apache version 2.0.

The project is split into several Maven artifacts. This approach decomposes problem into smaller parts which have dedicated functionality and can be easy reused in other projects. It also ensures that third party dependencies are loaded only for certain artifacts, where are needed.

Figure A.1 shows dependency tree of `datamining-dist`, which is the top level artifact. The most important modules are listed in Table 5.1. All ids of the listed artifacts have prefix `hawkular-datamining`.

Artifact Id	Description
`parent`	Manages shared dependencies and versions.
`forecast`	Core library for time series modeling and forecasting.
`api`	API used within Data Mining.
`cdi`	Support for context dependency injection.
`rest`	Web archive for standalone usage without Hawkular.
`dist`	Web archive with Hawkular integration code.
`itest`	Artifact for integration tests.

Table 5.1: Hawkular Data Mining modules.

In the following sections the most important parts of the implementation are described. This text focuses on design of data structures, interaction between them and algorithms.

5.1 Forecast Package

The forecast package is time series modelling and forecasting library. It contains several time series models and utility classes for time series modelling. It is designed to be easy used in any Java project. The

1. Available at `https://github.com/hawkular/hawkular-datamining`.

footprint of the artifact is small. It depends only on Apache Math, Commons, Guava and Jboss logging.

The core data structures from `datamining-forecast` package are interfaces of time series models and forecasters. Some of the methods are listed in Algorithm 5.1. These two interfaces are related but not the same. Each time series model should be capable of predicting and learning. Models also collect initialization and running statistics so it is possible to make assumptions of the forecasting accuracy.

The main difference between forecaster and time series model is that forecaster is designed for online learning and it autonomously selects the most appropriate time series model. Time series models by default are not designed for online learning, however it can be accomplished with a wrapper class.

Metric in the Data Mining is represented as structure `MetricContex` which contains metadata like collection interval, metric id and tenant id. Collection interval is necessary for calculating timestamps of predicted points.

Listing 5.1: Interface for time series models.

```
interface TimeSeriesModel {
    void learn(List<DataPoint> ts);
    List<DataPoint> forecast(int steps);
    int numberOfParameters();
    MetricContext context();
    AccuracyStatistics initStatistics();
    AccuracyStatistics runStatistics();
    ...
     }
```

List of implemented models, statistical and utility classes for time series manipulation:

- Simple ex. smoothing
- Double ex. smoothing
- Triple ex. smoothing
- Weighted moving average
- Automatic forecaster
- Augmented Dickey – Fuller test
- Time series decomposition
- Autocorrelation function

- Time series lagging
- Time series differencing
- Automatic period identification

5.1.1 Automatic Forecaster

One of the most important forecasting classes in Data Mining is `AutomaticForecaster`. This class autonomously decides which time series model should be used for modelled time series. Currently it decides from three implemented models: simple, double and triple exponential smoothing. Other models which implement interface `TimeSeriesModel` can be easily added.

Most importantly, this class is capable of dealing with concept drift. It holds circular buffer of historical metrics and if statistical properties of the underlying time series changes content of the buffer is used for selecting new model.

The strategy when a new model should be selected is configurable. The system currently supports two strategies. The first strategy selects new model periodically each n observations. The second one is more sophisticated and it selects a new model only when the error produced on learning data exceeds by x% error calculated on learning points when model was initialized.

Model selection is based on information criterion from Section 3.2. Automatic forecaster in successive steps calls optimizers of given models and then selects the best with the lowest information criterion. Used criterion is configurable for each forecaster.

5.1.2 Model Optimizers

Model optimizers are the most important part of the models implementation. If the model parameters are not estimated correctly, model produces high forecasting error.

The idea behind optimization is to find such parameters that describe training data the best [12]. Optimization criterion can be: mean squared error, mean absolute error and likelihood. These criterion basically computes errors produced by ahead predictions. The number of prediction steps depends on the forecaster's objective. The most common is to use only one step ahead. Data Mining currently supports only MSE criterion but others can be easy added.

The criterion is returned from model's objective function. This objective function is then passed to non-linear optimization algorithm from Apache Math Commons. This library implements several optimization algorithms: Nelder-Mead simplex, multi-directional simplex and bound optimization by quadratic approximation (BOBYQA). These algorithms do not need computed derivatives of the cost function.

In terms of the lowest execution time and quality of the estimated parameters the best results were produced by Nelder-Mead simplex algorithm.

Table 4.1 contains comparison of optimizers execution times from `OpenForecast` and R implementation. This section adds execution time of Data Mining implementation. Complete execution times are listed in Table 5.2. Execution times of Data Mining's optimizers are in some cases faster than R. Quality of the produced forecasts is discussed in Chapter 6.

Model	**OpenForecast**	**R `forecast`**	**Data Mining**
Simple ex.	4.91 sec.	0.003 sec.	0.033 sec.
Double ex.	9.31 sec.	0.006 sec.	0.002 sec.
Triple ex.	6.45 sec.	0.209 sec.	0.023 sec.

Table 5.2: Execution time of parameters estimation for exponential smoothing models.

5.2 API Package

The next package is `api`. It depends on `forecast` and it eventually could depend on another library for data analysis, for instance a library for an outlier detection. In this package there is a code necessary for using Data Mining as application which analyzes multiple metrics, for example classes for accessing data analysis objects for given metric.

The interface `SubscriptionManager` was designed for accessing time series analysis objects. Some of its methods are listed in Algorithm 5.2. Implementation of this interface could store entities in database.

However, Data Mining directly does not use any database. Therefore `SubscriptionManager` holds all objects in memory. Internally it stores objects in a map where the key is tenant and value another map where the key is metric id and value the object for data analysis – `Subscription`.

Listing 5.2: Interface `SubscriptionManager`.

```
interface SubscriptionManager {
    void subscribe(Subscription, Owner);
    void unsubscribe(tenant, metric, Owner);
    Subscription model(tenant, metric);
    ...
    }
```

For subscribing it is necessary to specify the owner of the prediction configuration. It was specially added for integration with Inventory. It is more described in Section 5.4.

Object `Subscription` holds instance of `AutomaticForecaster` and it could also hold another data analysis objects. Its role is to delegate calls to the business logic objects.

Class diagram of the most important classes from `forecast` and `api` is depicted in Appendix A.2.

Some of the service classes like `SubscriptionManager` are used in several other classes and also multiple implementation could exist (in memory or database implementation). Therefore dependency injection design pattern was introduced. Used technology is Java context dependency injection (CDI). In order to keep API classes clean module `cdi` with CDI configuration was introduced. It directly depends on `api`, but does not add any other business logic.

5.3 REST Package

Package `rest` contains implementation of REST services provided by Data Mining. Implementation is compatible with standard JAX-RS.

Data Mining REST API is designed to operate on the forecasting engine. When the module is deployed into Hawkular main interaction with Data Mining goes through Inventory REST API and metric data are collected from JMS topic. However, this is not limited and Data Mining offers alternative REST endpoints which can be used

for example in standalone deployment.

REST API provides following capabilities:

- Operate on subscriptions – enable and disable prediction.
- Push learning data to the engine.
- Get predictions for any number of steps ahead.
- Configure automatic forecaster.
- Get information about model being used and statistics of the predictions.

Accurate and more comprehensive REST API documentation is available on Hawkular web page[2]. This documentation is automatically generated using Swagger framework.

5.4 Distribution Package – Integration into Hawkular

As it was mentioned before, the build produces two web archives, one for standalone usage and another for deployment into Hawkular. The second web archive adds extra functionality for interacting with the other Hawkular components.

In Hawkular all entities are stored in Inventory, therefore Data Mining uses this module for querying metric definitions and it adds some extra attributes necessary for predictions. Another module which is used by Data Mining is Hawkular Metrics. Diagram 5.1 shows Data Mining integration into Hawkular.

2. Available at `http://www.hawkular.org/`.

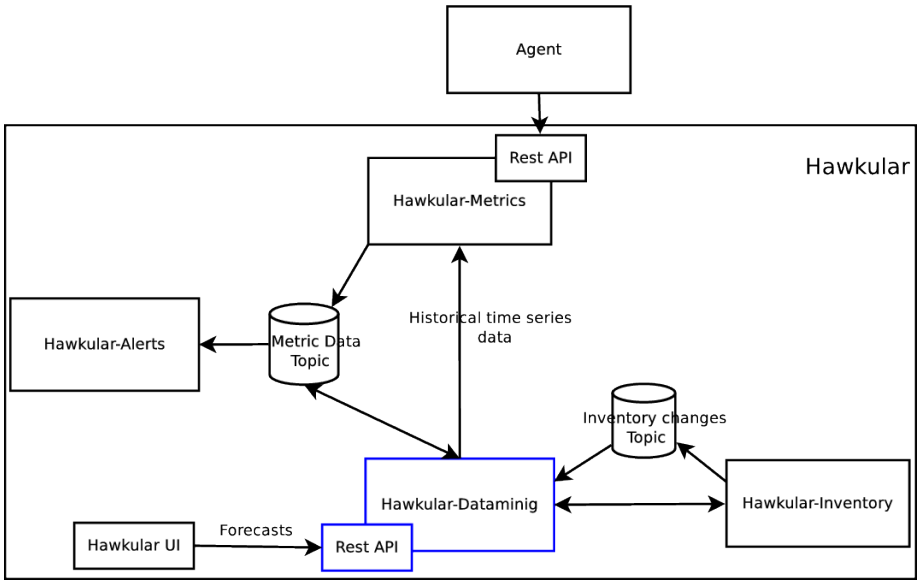

Figure 5.1: The integration into Hawkular.

When Data Mining is deployed into Hawkular, users can enable predictions by creating Relationship[3] from tenant to the target entity. Target entity can be metric, metric type or tenant. When relationship is created from tenant to tenant, predictions are enabled for all metrics under given tenant. Similarly, when a target entity is a metric type, predictions are enabled for all metrics of that type. In Figure 5.2 is depicted part of the structure of inventory with enabled predictions.

Relationships also store properties needed for predictions. One of them is forecasting horizon which tells how many prediction steps are performed for each incoming metric data point. With this approach of storing relationships in Inventory, Data Mining does not have to use any database for entities related to predictions. This approach was achieved without any changes to Inventory data model, this shows that Inventory's generic data model build on top of graph database[4] is good approach for storing data model in this domain.

3. Entity from Inventory which can be created between arbitrary two entities.
4. Compatible with Apache Tinkerpop.

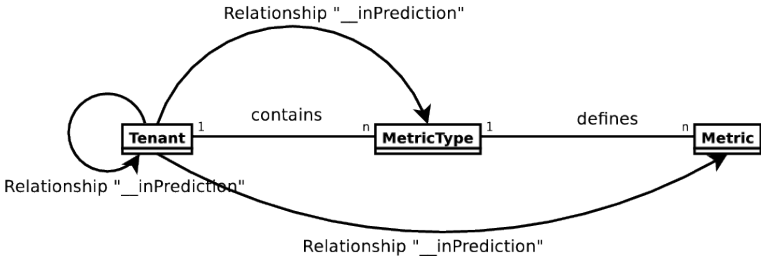

Figure 5.2: Relationship between entities of Inventory.

Any changes in Inventory are propagated to Data Mining by subscribing to specific events. This propagation is done through JMS topic where Inventory sends these events. However, on application start Data Mining also needs to query all predictive relationships from Inventory. For this purpose JMS request-response communication was implemented in Inventory. Any component which has access to internal messaging subsystem can construct a query and send it for execution to Inventory. REST API approach was also proposed but it could expose vulnerability – client could access any data objects of any tenant. Diagram 5.3 depicts sequence of calls when prediction gets enabled.

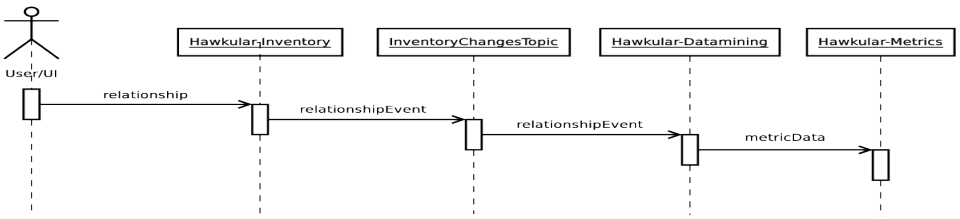

Figure 5.3: Sequence diagram of enabling predictions.

From Diagram 5.3 it is clear that when a prediction gets enabled, historical metrics are queried from Metrics module. This means that Data Mining needs to access metrics of all tenants. It is similar scenario to getting all predictive relationships from Inventory. It could not be implemented via REST API because of potential vulnerability and JMS based query was rejected by Metrics developers.

Another two solutions were proposed. The first was to bypass Metrics and query data directly from Cassandra. The second was to inject Metrics as CDI bean and directly use its Java API. The first solution was refused because the schema is maintained by Metrics and it can eventually change and also the backed database (Cassandra) can change. The final and implemented solution injectings Metrics as CDI bean. This approach assumes that Metrics are deployed in the same application server as Data Mining. From the performance perspective it is faster that JMS or REST calls.

Diagram 5.4 shows complete data flow from agent to alerts evaluation engine. First metric data is sent by agent to Metrics where it is stored to database. Then it is sent to JMS topic and consumed by Alerts and Data Mining. In parallel Alerts evaluates alerts criteria and Data Mining computes predictions. After computation the predicted data are sent back to the same JMS topic and consumed by Alerts. At this point an alert of predicted metrics can be triggered.

Data Mining changes `id` of predicted points in order to be able to distinguish original time series from predicted. For predicted metrics Alerts can evaluate the same conditions as for original time series or use different ones, more in Section 5.4.1.

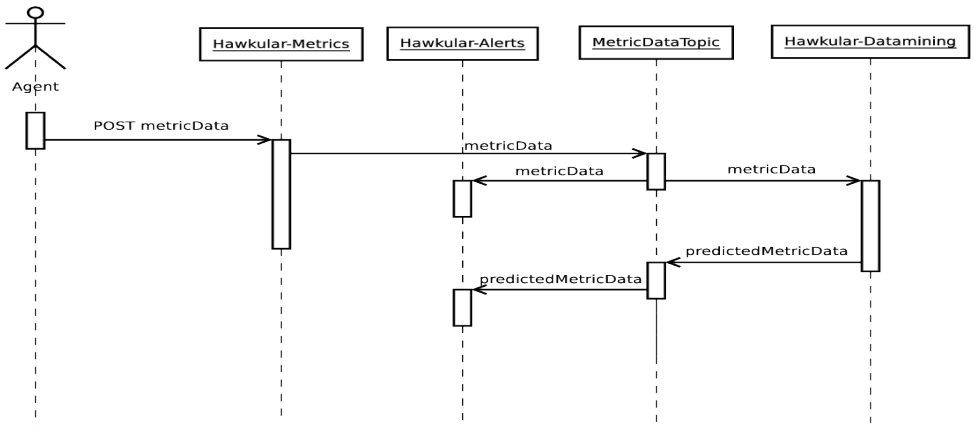

Figure 5.4: Sequence diagram of incoming data from agent.

5.4.1 Alerts and Conditions

In this section it is briefly described how the alerting system works. Alerts is a separate module which is responsible for firing alerts based on defined conditions. As it is depicted in Diagram 5.4 Alerts listens to the JMS topic and if conditions for given metric are positively evaluated an alert is fired.

Alerts data model is little bit more complicated though, as for each unique metric id multiple conditions can be defined. These conditions are evaluated for every received metric data point on the bus. Conditions are grouped under a trigger. If the result of evaluation is possitive than the tigger triggers an action. This action can be for instance sending an email, sms or perform a webhook...

Triggers can be grouped which facilitates creation of the similar triggers. Particulary this can be used for creating triggers for predictive metrics. If there is a trigger defined for the original metric it can be duplicated and used for predicted metrics.

If necessary the conditions can be changed in order to not trigger an alert if the predictions are not very accurate. Trigger dampening can be used for this. So for the trigger which operates on predicted metrics dampening can be set as follows: *"fire an alert if X consecutive evaluations of conditions are positive"*.

5.4.2 Hawkular UI

Predictive charts in Hawkular are located in Metric explorer tab. In Figure 5.5 predictive chart for metric accumulated garbage collection duration is shown. In this case automatic forecaster decided to use seasonal model of some automatically identified period. Note that for displaying original time series in the chart buckets were used, so the periods would be more easily seen on raw data points.

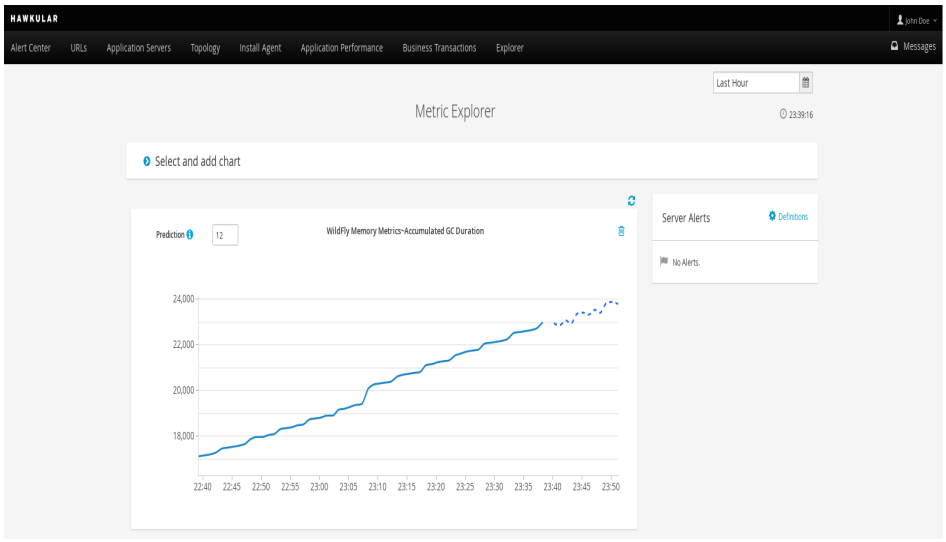

Figure 5.5: Metric explorer tab in Hawkular user interface.

After adding concrete chart predictions can be enabled by increasing prediction horizon input. So there is no need to enable predictions manually by creating relationship in Inventory.

Predictive charts were used from repository `hawkular-charts`. These charts are built on top of D3[5] library and encapsulated as Angular directives.

5.5 Test Automation and Documentation

Core functionality of the application is covered by unit tests. Junit framework is used for this type of tests. Because Data Mining interacts with other applications and modules, integration tests are also implemented. These tests are executed from Maven profile which starts web server with necessary modules deployed.

The last type of implemented tests are end-to-end tests which call REST endpoint of one module and expect result in call from another module. These types of tests are also executed in Maven profile with

5. Avalilable at `https://d3js.org/`.

running web server. End-to-end tests for Data Mining project live in the main Hawkular integration repository[6].

All tests are automatically executed every time developer creates a pull request or pushes code to the master branch. For this Travis continuous integration platform was used, which is available for free for open source projects on Github.

Documentation is directly written in Java code as javadoc. REST API documentation is also directly written in source code using Swagger framework and automatically generated at build time [7].

5.6 Releases

At the time of writing this thesis, there was only one release with version `0.1.Final`. It is also available in the Maven central repository. Complete group id is `org.hawkular.datamining`. The most important artifact ids are listed in Table 5.1. Release notes are available on Github[8].

5.7 Retrospective

In this section it is mentioned how the project was changing during the implementation, what did not worked well and what could have been done better.

At the beginning of the project Apache Spark[9] was used for building regression models. Its machine learning library (MLIB) offers several learning algorithms. In Data Mining linear regression with stochastic gradient descent was used. Because the gradient is stochastic, produced predictions are not very robust and quickly change by significant value.

Spark was mainly used because of its streaming extension that can handle high-throughput stream data [3]. There is also time series

6. Available at `https://github.com/hawkular/hawkular`.
7. Available at `http://www.hawkular.org/docs/rest/rest-datamining.html`.
8. Available at `https://github.com/hawkular/hawkular-datamining/releases`.
9. Open source cluster computing framework available at `https://spark.apache.org/`.

library `spark-ts`[10] which contains some time series models but does not offer functionality of automatic model selection.

Due to the problems with linear regression discussed in Section 2.2 and lack of automatic forecaster, decision was made to drop Spark and directly implement exponential smoothing models.

During the implementation a lot of time was spent on integration with other modules, with Inventory in particular. The problem was that Inventory is developed by another party and any concrete document describing the integration was not proposed. The design of the integration was understood differently by both parties, which lead to reimplementing of some parts. Therefore it is better to precisely define the integration interface with other modules or projects.

10. Available at `https://github.com/sryza/spark-timeseries`.

6 Evaluation of Implemented Models

Evaluation process compares predictive capabilities of the implemented models to statistics produced by similar models from R language. Models from R language are selected because they are widely used by forecasters all around the world and de facto standard for statistical science [13], therefore, for this work considered as referential.

To be specific used models are from package `forecast`[1]. Function used for fitting models is `ets`. This function accepts several configuration parameters for instance: what model should be used, statistics for objective function, information criterion... For testing in this work model parameter is set to `ANN`, `AAN` and `AAA` for simple, double and triple exponential smoothing, objective statistics for minimization is MSE and parameter damped set to false.

The used methodology and more comprehensive description of tested objectives is listed in Section 6.1.

6.1 Methodology

In order to evaluate forecasting quality of the implemented models it is necessary to use well defined methodology with test data which contains broad range of patterns: white noise, monotonicity, random shocks, trend and seasonality.

Most importantly methodology should not evaluate forecasting accuracy how well model fits the historical data. Instead it should evaluate forecast accuracy using genuine forecasts – test how well model performs on new data [10]. Therefore it is necessary to split time series into training and test set.

The size of the test set is usually 20% of the total sample. Albeit the size of test set depends on how long all the sample is and how far ahead one wants to forecast.

Outline of the methodology used in this work:

1. Divide data sample into training and test set.
2. Set $k = size(trainingSet)$.

1. Available at `https://cran.r-project.org/web/packages/forecast/index.html`.

3. Train model on the training set. And learn up to k^{th} observation.

4. Compute n-step ahead forecast $\hat{y}$ and error $\epsilon = y - \hat{y}$.

5. Set $k = k + 1$ and continue at step 3 until $k = size(trainingSet + testSet)$.

6. Calculate MSE of produced errors.

As it was described earlier time series used for testing should contain one or even combine more patterns together. Test samples used in this work are depicted in Appendix C. Each sample contains white noise, random shocks and one or more of the following patterns: monotonicity, trend and seasonality. Names and description of test samples are listed in Table 6.1.

Time Series Name	**Description**
`wn`	Constant white noise.
`trendUpLow`	Upward trend with white noise.
`trendDownHigh`	Downward trend of high variance.
`sine`	Constant sine with white nosie.
`sineTrend`	Sine with trend and white noise.

Table 6.1: Description of test samples.

The last part what figures in our methodology is training size. Let's say that one would like to forecast one hour ahead and Wildfly Agent collects most of the metrics in five minute intervals. Thus forecasting horizon of one hour corespondes to 12 observations and training set should be at least long 48 observations.

Interesting is also to find out if forecasting accuracy changes for larger training set. Therefore it was decided to add training sets of length 100 and 200 observations.

In the following sections each model is tested using methodology from Section 6.1 on samples from Table 6.1 of training size of 48, 100 and 200 observations with forecasting horizon set to: 1, 6 and 12 steps ahead. The size of test set is always 12 observations.

Following notation is used to identify test cases:

- *trainSampleSize – testSampleSize (forecastingHorizon)*

So for instance 48 – 12(1) denotes: training sample of size 48 observations, 12 test observations and forecasting horizon 1 step ahead.

6.2 Results

Complete test results are listed in Appendix D. The top value in each cell is the result from R and below is the result from Data Mining. All measured values are in mean squared error. The table contains lot of data, however this work focuses on answering these two questions:

1. *How well do the Data Mining models perform compared to R models?*

2. *Does the size of training set affect quality of the forecasts?*

The first question can be answered by calculating ratios of Data Mining MSE to R MSE. If the ratio is lower than one it means that Data Mining model forecast more accurately.

From the test results it can be shown how many tests cases have lower ratio or even better it is to split ratios into following intervals: lower than 1, [1 , 1.01] , [1.01 , 1.05] , [1.05 , 1.10] , [1.1 , 1.2] and higher than 1.2. More aggregated representation is to calculate averages of these ratios for each test case.

In order to get the answer for the second question it is necessary to compare MSE statistics produced by Data Mining models. For instance compare the result of test case 48 – 12(1) on `wn` sample to the same test case but with longer training set and analogously for other test samples and longer forecasting horizons. With this approach there would be too many results to compare. So this work first calculates averages of mean squared error of each test case, so the average for 48 – 12(1) is the average of all test samples (`wn`, `trendUpLow...`) of given test case.

Chart with clear results is constructed for each of these two questions. The results of these charts are described in the following sections.

6.2.1 Simple Exponential Smoothing

Raw results for simple exponential smoothing are listed in Table D.1. Pie Chart 6.1 shows that Data Mining simple exponential smoothing model produces lower MSE than R in 38% of all forty-five test cases and 38% tests results are equal or worse at mosts than 1%. The test with worst result has 8% higher MSE than equivalent R model.

Data Mining's best result has 23% lower MSE than the result from R model. It is for test case 100 – 12(1) on test sample `trendUpLow`.

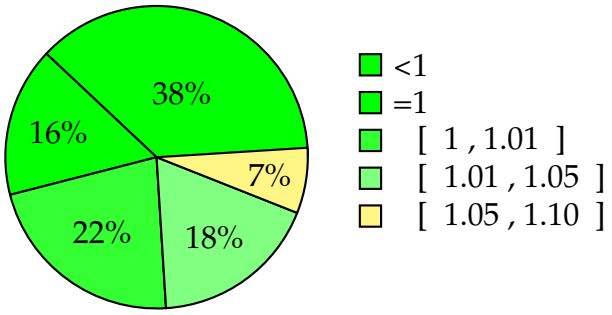

Figure 6.1: The proportion of Data Mining MSE to R MSE ratios split into intervals.

Chart 6.2 shows average ratios of Data Mining MSE to R MSE for each test case. These results are finer variants of results from Chart 6.1. It highlights that only three test cases out of nine have in average higher MSE than R and at most 2% higher. It also shows that for larger training set Data Mining simple exponential smoothing in average produces lower MSE than R and on shorter training set R produces lower MSE than Data Mining.

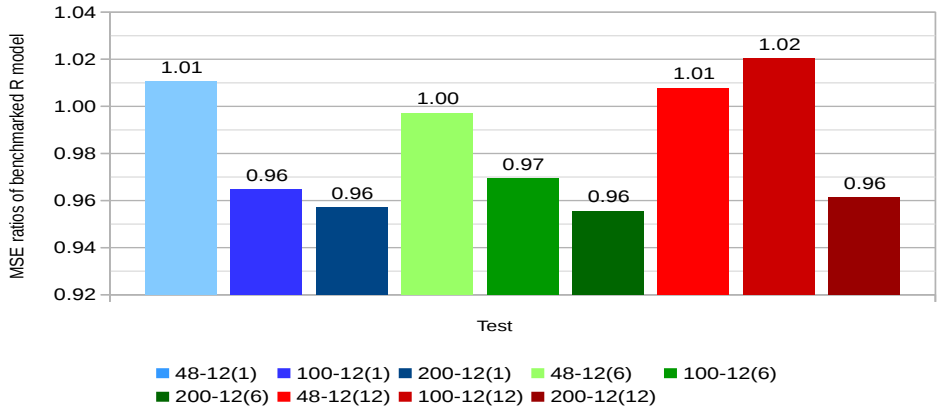

Figure 6.2: Test results for triple exponential smoothing. Comparison to reference R model.

Averages for each test case of Data Mining MSE are listed in Chart 6.3. This chart shows that the lowest average MSE are produced in test cases with training size of 48 observations. Therefore for simple exponential smoothing it is not necessary to use longer training set in order to get lower prediction error. Note that training samples contains randomly generated shocks. Therefore if more shocks are located later in time series than test cases for larger training sample can have higher MSE.

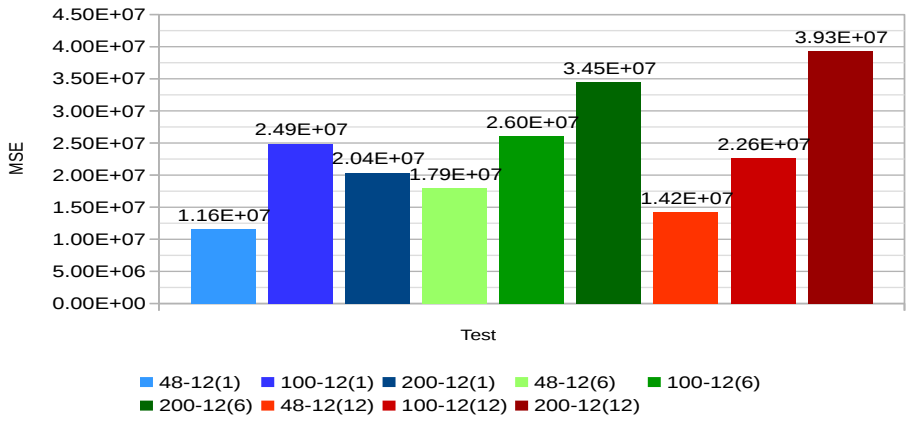

Figure 6.3: Test results for simple exponential smoothing. Average values of MSE for all tests samples.

6.2.2 Double Exponential Smoothing

Complete raw tests results for double exponential smoothing are listed in Table D.2. Chart 6.4 emphasizes that in 51% tests Data Mining produces lower MSE than the equivalent R model. Five tests fall into the interval $[\ 1.1\ ,\ 1.2\ ]$ and one test has higher MSE than 20%. Best result is for test case $100-12(12)$ where Data Mining produces lower MSE of 13%.

Averages of MSE ratios are depicted in Chart 6.5. Only three test cases have higher MSE than 1% to maximum 6%. The chart also shows that in average the lowest MSE ratios are for training set of length 48 observations. It meas that if results are compared to R in terms of training size R produces higher mean square error for larger training sets.

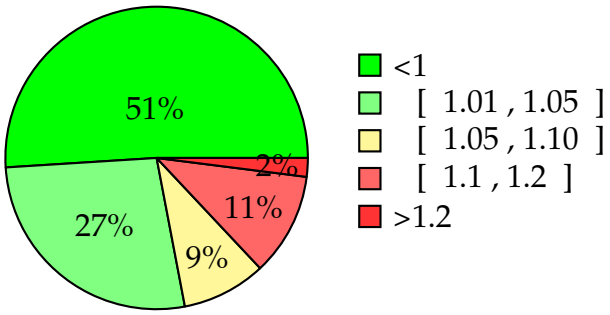

Figure 6.4: The proportion of Data Mining MSE to R MSE ratios split into intervals.

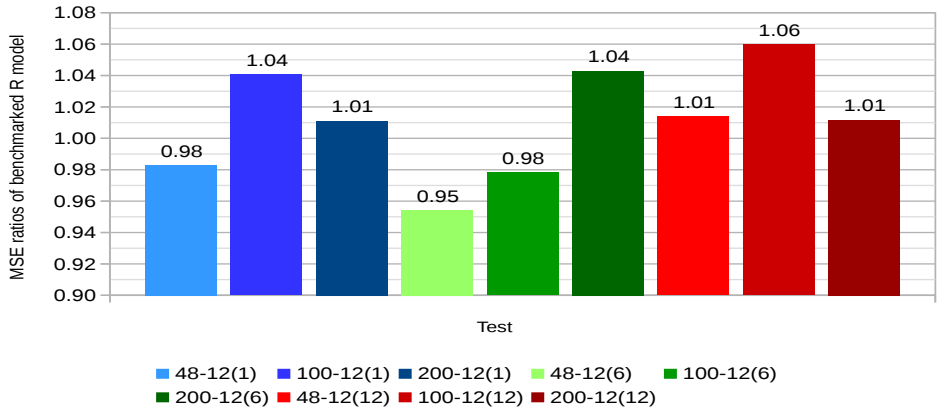

Figure 6.5: Test results for double exponential smoothing. Comparison to reference R model.

The last chart for double exponential smoothing is 6.6. It shows that model similarly as simple exponential smoothing produces higher MSE for larger training sets.

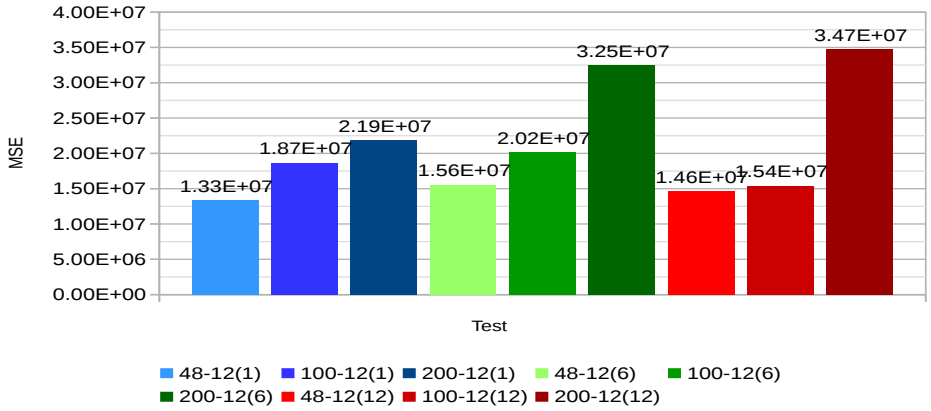

Figure 6.6: Test results for double exponential smoothing. Average values of MSE for all tests samples.

6.2.3 Triple Exponential Smoothing

Raw tests results for triple exponential smoothing are listed in Table D.3. In this case test cases can be run only on two data sets, because it does not make sense to fit seasonal model on non seasonal data. Therefore in total only eighteen tests were performed.

During the testing it was observed that execution time of R script takes significantly more time than for other exponential smoothing models. On the other hand Data Mining implementation for triple exponential smoothing does not take more time compared to other exponential smoothing models. Table 5.2 shows that R in average for one estimation takes 0.209 seconds which is for eighteen executions equals to 3.8 seconds. Whereas one estimation in Data Mining takes in average 0.023 seconds so time for eighteen estimation is 0.4 seconds.

Pie Chart 6.7 shows that Data Mining triple exponential smoothing produces lower MSE in 67% of all eighteen tests. It is better result than for simple or double exponential smoothing. However three tests produced MSE higher than 20%. This could happen for example because R may use some better strategy for decomposing time series and setting seasonal indices.

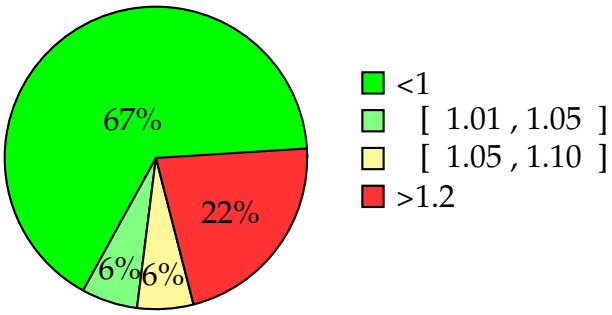

Figure 6.7: The proportion of Data Mining MSE to R MSE ratios split into intervals.

Averages of the MSE ratios are showed in Chart 6.8. From the chart it can be seen that four out of nine tests produces higher MSE. R produces only 0.95 MSE in test case 200 – 12(12) on `sine` sample, but for smaller training sizes it produces 4.31 and 2.26 which is quite unstable result. Data Mining produces 4.07, 2.15 and 2.63 (training sets of size 50, 100 and 200). This causes significantly higher ratio for this test case.

Overall results for averaged MSE ratios show that Data Mining performs better than R on smaller training sets.

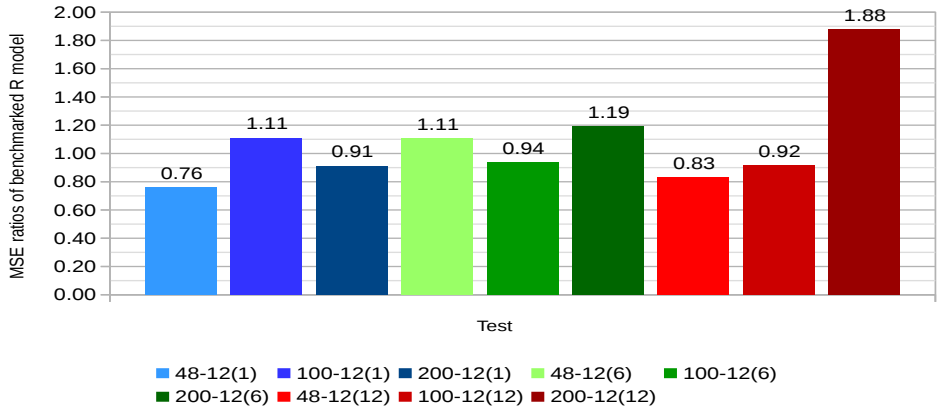

Figure 6.8: Test results for triple exponential smoothing. Comparison to reference R model.

Chart 6.9 emphasize that larger training size does improve forecasting accuracy. It is the same result as for other models.

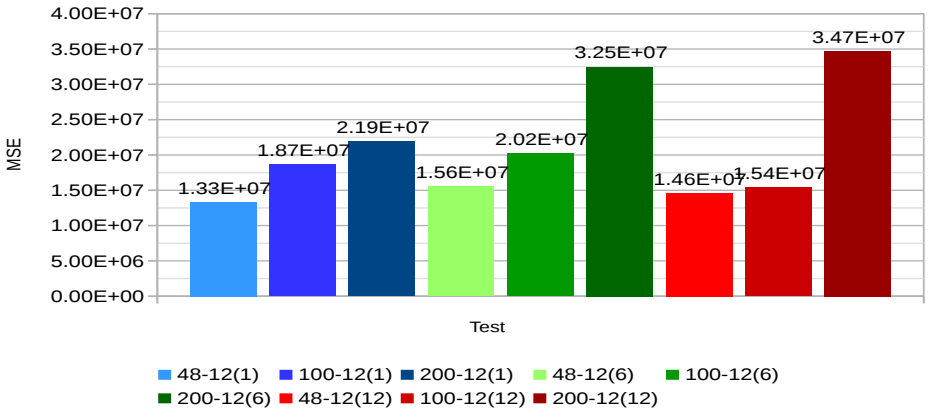

Figure 6.9: Test results for triple exponential smoothing. Average values of MSE for all tests samples.

6.2.4 Overall Results

This section contains aggregated results for simple, double and triple exponential smoothing models.

Chart 6.10 highlight that only in four tests Data Mining models in average produces higher MSE than alternative R models. Test case 200 – 12(12) shows the worst result which is 28% higher than benchmarked R model. This was discussed in Section 6.2.3.

Average for all test cases is equal to 2%. In other words Data Mining models in average produced mean squared error higher than 2% compared to R models. Therefore Data Mining model's prediction capabilities can be considered as R alternatives.

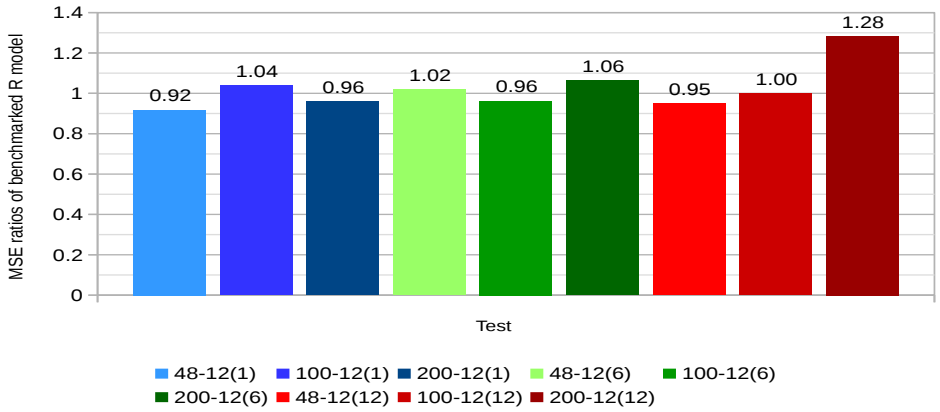

Figure 6.10: Aggregated test results for all models. Comparison to reference R model.

Aggregated results for Data Mining mean squared errors are depicted in Chart 6.11. The results are similar to the results discussed for each model. It shows that longer training set does not improve forecasting accuracy. Further research could identify ideal size of training set. These tests proved that longer training size does not imply better forecasts.

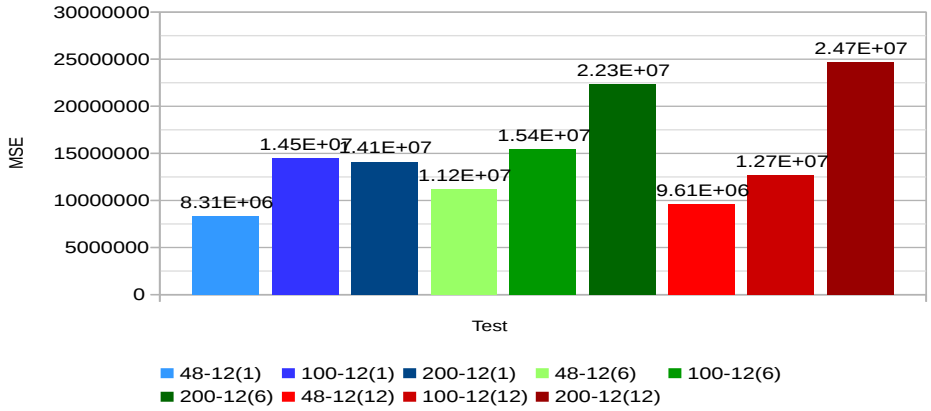

Figure 6.11: Aggregated test results for all models. Average values of MSE for all tests samples.

7 Conclusion

The goal of the master's thesis is to develop independent module in Java language for providing alert prediction capabilities for open source project Hawkular.

Implementation part is split into multiple artifacts with dedicated functionality. The most important module focuses on time series modelling. It contains simple, double and triple exponential smoothing models which use non-linear optimization algorithm for finding the best parameters with goal of minimizing prediction error. The library also contains several time series utility classes and statistical tests for time series analysis.

Data Mining prediction engine autonomously selects the best time series model for underlying time series. Most importantly, it is capable of dealing with concept drift problem. So there is no need of any analytical configuration in production.

The last chapter benchmarks forecasting accuracy of the implemented models. It is shown that forecasts produced by implemented models are in some cases more accurate that forecast from package `forecast` from R language and in average Data Mining models can be compared to R models. In terms of execution time performance Data Mining models are much efficient.

During the development of the module, Hawkular was still in development phase, therefore it was not possible to gather real data from production environment. This would help with tunning models for specific metrics for example Java heap usage.

The module was successfully integrated into Hawkular and predictive charts of the metrics are located in Metric explorer tab.

Future work on this module should analyze data from production servers and if necessary change the properties of optimizers in order to get more accurate predictions. Analysis could also contain evaluation if damped trend should be added to the implemented models.

As it is open source project, it would be great to attract more people into development and continue with adding new models and features for time series analysis, because there is no widely used and maintained library for this purpose in Java.

Bibliography

[1] *IBM Operations Analytics – Predictive Insights* [online]. [cit. 2016-04-20]. Available at: <https://www-03.ibm.com/software/products/en/ibm-operations-analytics---predictive-insights>.

[2] *Measuring time series characteristics: Finding the period of the data* [online]. [cit. 2016-4-12]. Available at: <http://robjhyndman.com/hyndsight/tscharacteristics>.

[3] *Spark Streaming Programming Guide: Overview* [online]. [cit. 2016-4-29]. Available at: <https://spark.apache.org/docs/latest>.

[4] *New Relic* [online]. 2016 [cit. 2016-4-20]. Available at: <https://en.wikipedia.org/wiki/New_Relic>.

[5] *Root cause analysis of infrastructure issues* [online]. 2016. 2016-4-11. Available at: <https://help.ruxit.com>.

[6] Aras, S. a Deveci İpek. A new model selection strategy in time series forecasting with artificial neural networks: IHTS. *Neurocomputing*. 2016, roč. 174. S. 974–987.

[7] Brockwell, P. a Davis, R. *Time Series: Theory and Methods*. Praha: Springer-Verlag New York, 2009. ISBN 978-0-387-97429-3.

[8] COCIANU, C.-L. a GRIGORYAN, H. An Artificial Neural Network for Data Forecasting Purposes. *Informatica Economica*. 2015, roč. 19, č. 2. S. 34–45. ISSN 14531305.

[9] Cyhelský, L. a Souček, E. *Základy statistiky*. Praha: Vysoká škola finanční a správní, 2009. ISBN 978-80-7408-013-5.

[10] Hyndman, R. a Athanasopoulos, G. *Forecasting: principles and practice: Time series components* [online]. [cit. 2015-1-16]. Available at: <https://www.otexts.org/book/fpp>.

[11] Hyndman, R., Koehler, A. B., Ord, J. K. et al. *Forecasting with exponential smoothing: the state space approach*. [b.m.]: Springer Science & Business Media, 2008.

[12] Hyndman, R. J., Koehler, A. B., Snyder, R. D. et al. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*. 2002, roč. 18, č. 3. S. 439–454.

[13] Kleiber, C. a Zeileis, A. *Applied econometrics with R*. [b.m.]: Springer Science & Business Media, 2008.

[14] Sang, Y.-F., Wang, Z. a Liu, C. Period identification in hydrologic time series using empirical mode decomposition and maximum entropy spectral analysis. *Journal of Hydrology*. 2012, roč. 424. S. 154–164.

[15] Tomáš, C. *Finanční ekonometrie*. [b.m.]: Ekopress, 2008. ISBN 978-80-86929-43-9.

[16] Wikipedia. *Moving-average model* [online]. 2016 [cit. 2016-4-11]. Available at: <https://en.wikipedia.org/wiki/Moving-average_model>.

[17] Zhang, G., Eddy Patuwo, B. a Y Hu, M. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*. 1998, roč. 14, č. 1. S. 35–62.

A Diagrams

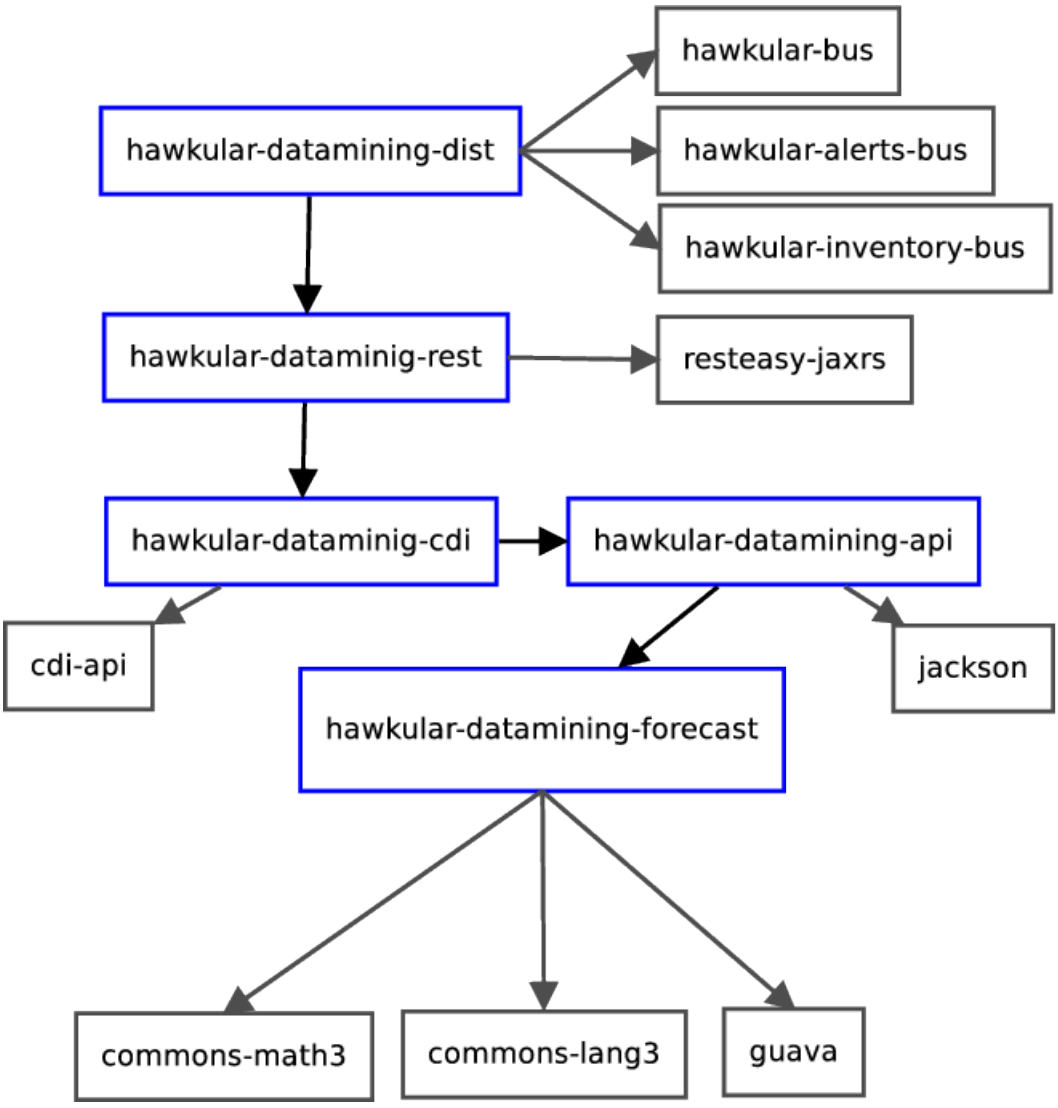

Figure A.1: Dependency tree of maven artifacts.

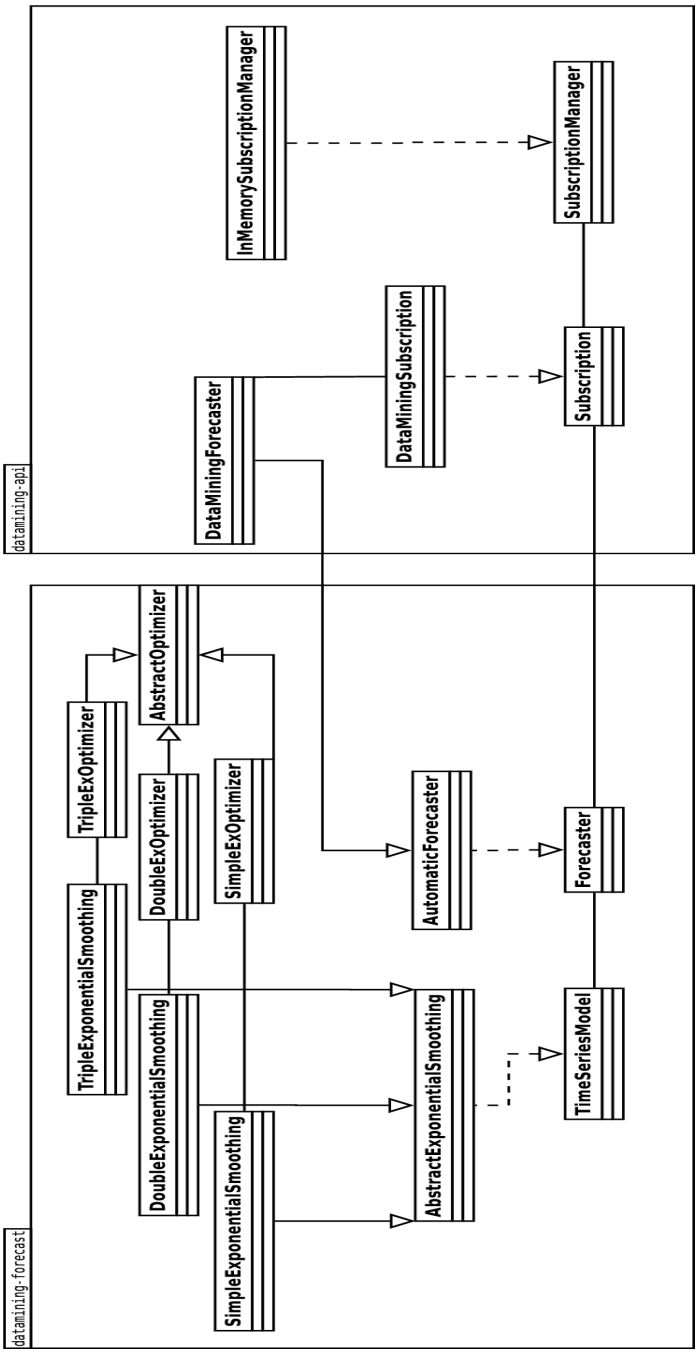

Figure A.2: Partial class diagram of `datamining-forecast` and `datamining-api`.

B Predictive Charts

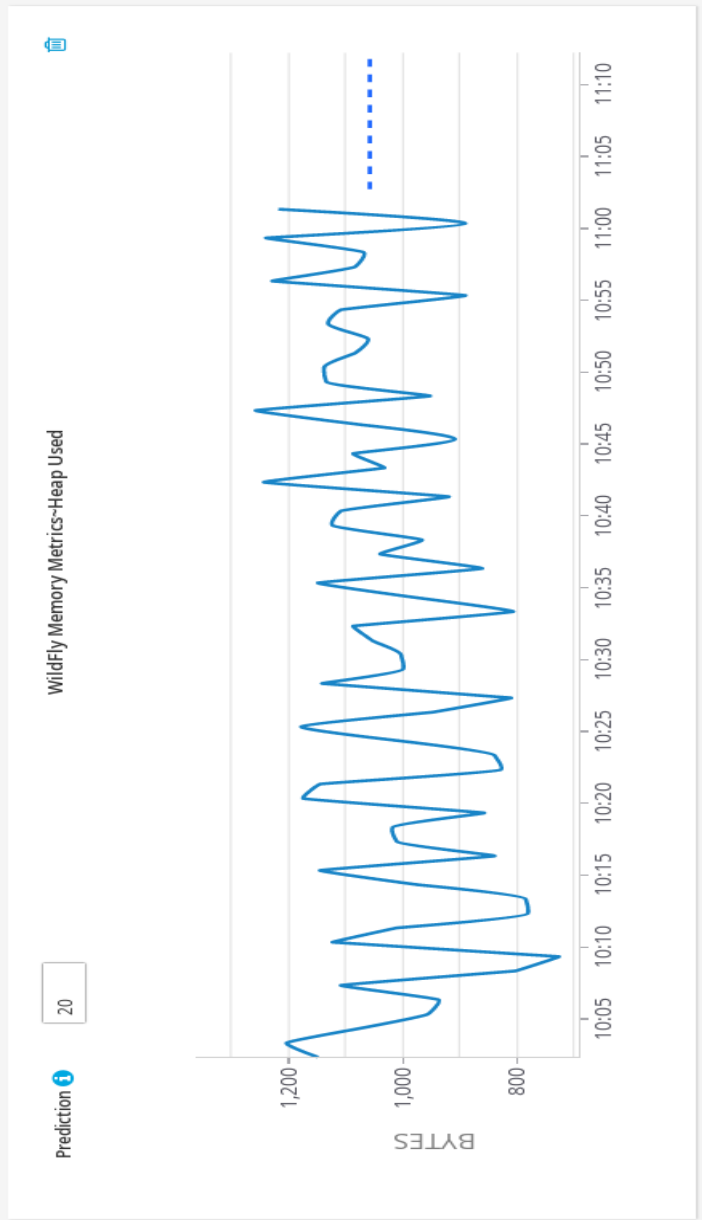

Figure B.1: Predictive chart for simple exponential smotohing.

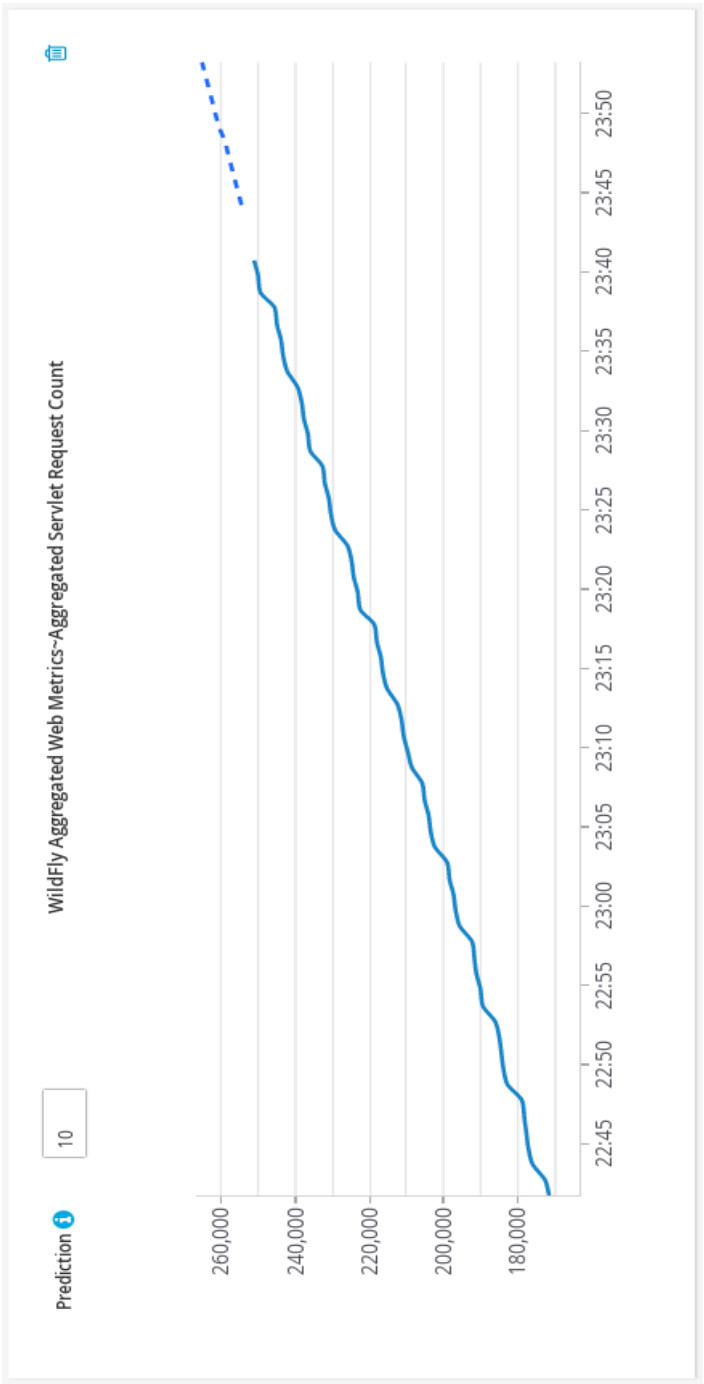

Figure B.2: Predictive chart for double exponential smotohing.

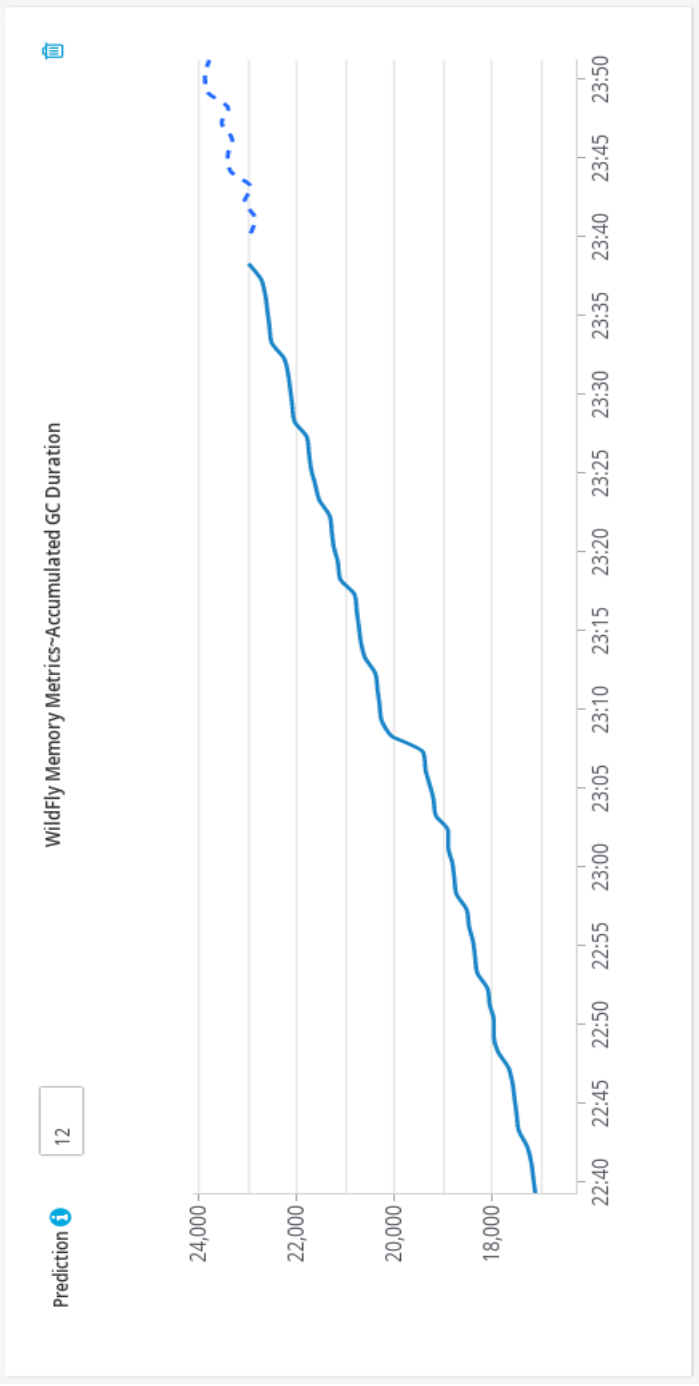

Figure B.3: Predictive chart for triple exponential smotohing.

C Testing Time Series Samples

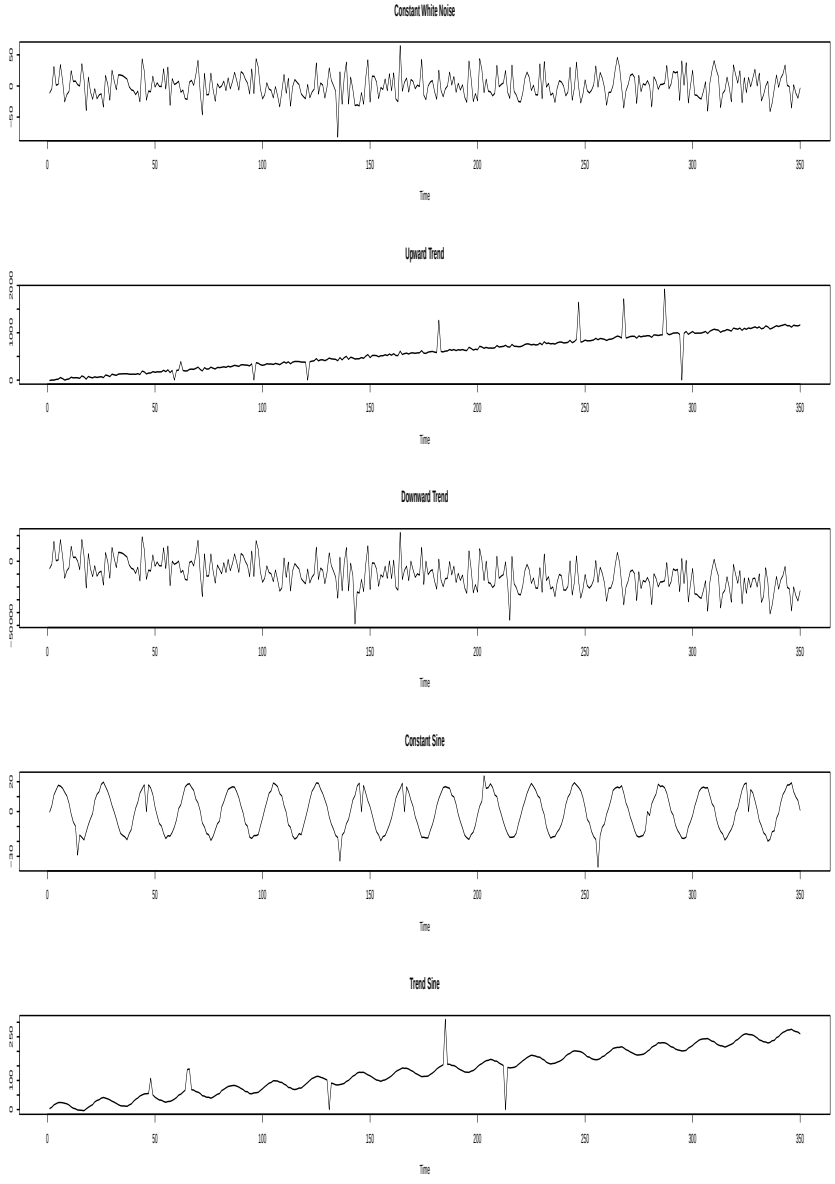

Figure C.1: Time series samples for evaluation.

D Tests Results

Test case	wn	trendUpLow	trendDownHigh	sine	sineTrend
48 – 12 (1)	252.60	4293.10	59074630.35	19.91	291.89
	251.72	4602.83	57931198.55	19.97	291.90
100 – 12 (1)	247.72	702.52	117983966.67	23.54	25.82
	252.96	513.46	124329179.62	23.54	26.30
200 – 12 (1)	380.51	862.58	100868036.49	45.14	50.91
	380.57	718.39	101969913.50	45.14	47.95
48 – 12 (6)	333.07	7071.66	90579713.93	483.70	1772.85
	325.33	7206.03	89687108.81	483.88	1772.86
100 – 12 (6)	287.00	1916.31	125727085.38	420.33	346.06
	289.63	1531.48	130162180.73	420.33	347.18
200 – 12 (6)	325.67	1457.51	172882341.31	394.90	2734.56
	327.13	1132.87	172309610.96	394.90	2733.64
48 – 12 (12)	271.78	7556.76	71166890.64	535.57	2956.52
	276.16	7742.00	71094737.26	535.46	2956.52
100 – 12 (12)	239.68	12726.82	104525954.16	513.42	288.09
	251.44	12416.04	112937185.50	513.42	286.96
200 – 12 (12)	381.57	2875.74	197649156.81	562.27	2064.04
	381.71	2316.18	196658315.76	562.27	2077.42

Table D.1: Test results in MSE for simple exponential smoothing.

Test case	wn	trendUpLow	trendDownHigh	sine	sineTrend
48 – 12 (1)	272.33	3617.53	65694458.22	22.70	668.61
	279.60	3574.63	66644587.71	20.41	659.66
100 – 12 (1)	282.50	190.63	90967195.65	8.46	26.17
	302.91	193.77	93267732.29	9.40	25.61
200 – 12 (1)	399.27	361.50	108286781.11	35.56	48.96
	405.67	347.69	109496254.07	38.48	48.21
48 – 12 (6)	354.67	6324.32	86541840.59	880.76	8031.60
	320.00	6454.35	77856708.99	856.96	7843.41
100 – 12 (6)	323.09	263.81	110815056.99	636.99	449.11
	332.08	253.23	100878861.50	673.94	420.09
200 – 12 (6)	324.50	316.76	167489984.23	811.24	2958.53
	340.14	374.38	162368879.38	830.81	2933.63
48 – 12 (12)	275.92	3077.24	65058787.15	4394.80	29063.69
	299.00	3068.45	72999868.22	3921.33	28324.64
100 – 12 (12)	272.97	13143.67	58829949.01	4919.03	565.68
	315.49	13449.97	76982996.39	4646.99	490.45
200 – 12 (12)	376.87	423.64	186182717.66	5777.67	2507.91
	381.13	499.08	173511817.12	5462.19	2489.15

Table D.2: Test results in MSE for double exponential smoothing.

Test case	sine	sineTrend
48 – 12 (1)	11.24	25.08
	8.07	20.15
100 – 12 (1)	2.29	43.64
	2.09	57.07
200 – 12 (1)	15.92	88.18
	15.74	73.84
48 – 12 (6)	9.26	390.64
	4.07	693.93
100 – 12 (6)	3.09	34.05
	2.44	37.11
200 – 12 (6)	4.19	1853.59
	5.65	1908.62
48 – 12 (12)	4.31	965.75
	4.07	693.93
100 – 12 (12)	2.26	8.44
	2.15	7.48
200 – 12 (12)	0.95	1954.75
	2.63	1924.38

Table D.3: Test results in MSE for triple exponential smoothing.

E Source Code Metrics

Number of Java files: 102

Number of lines: 9289

Size of the WAR archive of `hawkular-dataminig-dist` module : 8744B

F Content of the Attachment

Directory `doc`: LaTeX source code of this thesis.

Directory `src`: source code of Data Mining module.

Directory `src-hawkular`: source code of Hawkular with integrated Data Mining.

Directory `bin`: binaries of Data Mining module and Hawkular with integrated Data Mining.