Masarykova univerzita
Fakulta informatiky

Alert Prediction in Metric Data Based on Time Series Analysis

Master thesis

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.

Pavol Loffay

Advisor: RNDr. Adam Rambousek, Ph.D.

Acknowledgement

I would like to thank my supervisor Jiri Kremser, family and the people from Hawkular team. Next, my thanks goes to Ing. Daniel Němec, Ph.D. and Ing. Daniel Schwarz, 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 the 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.

Keywords

Time Series, Hawkular, Alert Prediction

Contents

1 Introduction

In driving successful business on the internet it is important to assure an application health and reliability. One can achieve that by monitoring subjected resources and setting up a clever alerting system. These features are offered in many monitoring systems, however being predictive in this area can even prevent undesirable states and most importantly gives administrators more time for reacting to such events. For instance it can decrease downtime of an application or ability to load balance workload in advance by horizontal scaling targeted services.

As alerting system are sophisticated and can be composed by many conditions so this work focuses only on predicting future metrics values which are then sent as input to an alerting system.

In the first chapter are discussed various approaches for time series modeling and forecasting. Second chapter focuses on implemented models with validation on real and generated test data sets. Implementation details with testing can be found in fourth chapter.

TODO describe chapters

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 architecture and used technologies had to fit into the overall project architecture.

Hawkular is middleware monitoring and management platform developed by company Red Hat and independent community of contributors. It is a successor to very successful RHQ[2] project, also known as JBoss Operations Network (JON). By monitoring 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 RHQ project was due it's size hard to maintain and lacking robust REST api lead to fresh development

1. Available at `http://www.hawkular.org`
2. Available at `https://rhq-project.github.io/rhq/`

of new application. In contrast Hawkular consist of several loosely coupled or even independent applications. These independent components are much easier to maintain and more importantly 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 product is customized Wildfly [3] application server with all components deployed in it.

- 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 uses also Java messaging topics (JMS) for inter – component one to many communication.

Modules are packaged as standard Java web archives (WAR), or enterprise archives (EAR) and deployed into customized 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 module for Hawkular which will provide forecasts for any time series metrics collected by agent. On new metric data available the module learns from data and predicts new values. Based on this predicted values an alert can be triggered. Forecast should be also available for user interface in predictive charts.

3. An open source project of JBoss EnterpriseApplication 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.

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 server each with multiple agents. Therefore performance of chosen learning algorithm has to be taken in account.

2 Time Series Models

This chapter focuses on time series theory and various approaches for modelling time series. Models are ordered from simpler to more complex ones.

Firstly, it is important to define time series; it is sequence of observations $s_t \in \mathbb{R}$ ordered in time. This thesis focuses only on univariate equidistant discrete time series. Time series analysis contains many segments, this work focuses on forecasting. 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.

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.

TODO mention that we talk only about models which makes sense to use in our environment.

2.1 Simple quantitative methods

Following methods are the simples forecasting quantitative models. They can be used on any time series without 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.

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

There is also a seasonal variant of naïve model. This method is suitable only for highly seasonal data. These methods in general produces high forecasting error but are very easy to implement.

2.2 Linear Regression

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

Simple linear regression is defined as:

$$y = \beta_0 + \beta_1 x + \epsilon \quad (2.4)$$

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

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

Once parameters are estimated predictions for any time in the future can be calculated. If modelled time series is not stationary then and for instance trend changes over time parameters has to be periodically estimated to achieve better accuracy.

2.3 Simple Exponential Smoothing

The concept behind simple exponential smoothing is to attach larger weights to the most recent observations than to observations from distant past. Forecasts are calculated using weighted averages where

the weights decrease exponentially as observations come from further in the past. In other words smaller weights are associated to 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 observations from distance in past.

Simple exponential smoothing has flat forecast function, that means all forecasts all the same. Smoothing can be generally used as technique to separate signal and noise. 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 can produce high error.

2.4 Holt's Liner Trend Method

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 original one without trend. In order to add trend component another equation has to be added.

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

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

2.5 Holt – Winters Seasonal Method

This method is an extension of Holt's linear trend method with added seasonality. It is also called triple exponential smoothing. In this model there are three equations 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.

2.6 Box – Jenkins Methodology (ARIMA)

Models from Box – Jenkins methodology are the most widely used in time series analysis specially 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 together autoregressive, integrated and moving average models. An autoregressive model (AR) consist of sum of weighed lagged observations. It is listed in 2.9. The order of this model is defined by p and can be determined from PACF function [9]. A moving average model (MA) is sum of weighted errors of order q. The order of this part can be determined from ACF function [9]. 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. The order if differencing original series is denoted by d letter. Usually first order differences are enough to make time series stationary.

Parameters of the model, including AR and MA part can by estimated by non-linear least squares or maximum likelyhood estimation [5]. For successful estimation a certain number of historical points needs to be available. In [9] minimal training size is set to at least fifty observations.

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

Moving averages model should not be confused with simple moving average 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 [10]. 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.

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

It is important 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 [5]. 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).

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

After this theoretical part it is clear that ARIMA models are more complicated than family of moving averages.

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 [11]. 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 [4]. 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 [6].

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 [7]. 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 it 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 and multiplicative 2.12. 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 consist of following steps:

- Compute a trend component $\hat{T}_t$ using moving average model. If a period is even use $2xMA(period)$. If period is an odd number use $MA(period)$. $2xMA$ for even period is used because it has to be symmetric.
- Calculate detrended series $y_t - \hat{T}_t$.
- 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.
- 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 test are used whether a time series in non stationary. In this work Augmented Dickey – Fuller (ADF) test was chosen for unit root testing. It's null hypothesis H_0 is 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 some parts of equation 2.14. The most important ones and widely used are:

- *nc* – no constant - for regression with no constant not time trend (βt)

- c – constant for regression with an intercept but no time trend (βt)

- ct – for a regression with an intercept and time trend

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

The implementation of this test is to fit multiple linear regression model of equation 2.14. Then calculate ADF statistics with 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 of 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, specially if there is significant noise present in the series [8].

There are several ways how to implement automatic period identification. The most used ones are based on autocorrelation function (ACF) or spectral density [1]. 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 is decreasing to zero.

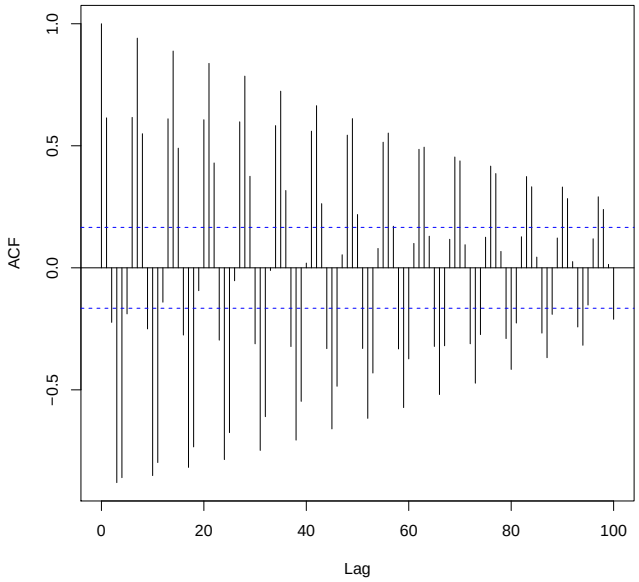

Figure 2.1: ACF of sine function.

The algorithm for automatic period identification is demonstrated in 1. It uses ACF function It is looking for periodically significant values of ACF function. These values have to be decreasing to zero at some rate. At the infinity lag ACF approach zero.

The algorithm first calculates autocorrelation function of input time series and it finds index of the highest value. Follows `checkPeriodExists` where it checks if there are significant values of ACF present at following $n * period$ indices of autocorrelation function. 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:      acf ← acf(ts)
5:                                  ▷ returns index of the highest value
6:      period ← findHighest(ts, period)
7:      while period ∗ 2 < ts.length do
8:          if checkPeriodExists(x, ts) then
9:              return period
            period ← findHighest(ts, period)
10:     return 0
```

3 Models Demonstration

In the previous chapter several models for forecasting were discussed, however in Hawkular only a few of them were selected and implemented. It is because various time complexity of the models and more important robustness in terms of beeing able to produce accurate results for higher range of modelled time series. Following model evaluations and graphs are generated using statistical system R.

3.1 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 is monotonically decreasing or increasing and availability represents up or down state of a resource.

3.2 Evaluating Forecasting Accuracy

In order to evaluate model it is important to estimate an error of the forecast. There are several statistics for evaluating forecasting errors. The most used ones are mean squared error (MSE) 3.1a and mean absolute error (MAE) 3.1b. The difference between them is that MSE emphasizes the extremes while MAE is more robust to outliers.

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

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{3.1b}$$

3.3 Model Selection

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 overfits training data and most importantly it selects less robust model [9]. Therefore for comparing multiple models another factors has 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.

$$\begin{aligned} AIC &= 2k - \ln(L) \\ BIC &= k\ln(n) - 2\ln(L) \end{aligned} \tag{3.2}$$

Equations for AIC and BIC are listed in 3.2. Number of parameters of the model represents k, n is number of observations and L is 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 3.3. Corrected version of AIC more penalizes longer models.

$$AICc = AIC + \frac{2k(k+1)}{n-k-1} \tag{3.3}$$

3.4 Evaluation on Data

An analytical process of modelling time series advise to plot the data and fit various models chosen by an forecaster previous experience. In second step compare statistics of those models and choose one the best describes data. However, this approach can be used only if a process of modelling is conducted by human being.

In this section a real time series is modelled using time series discussed in section 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.

In 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 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 one step lag. These two models differs in forecasts. Drift model is capable of capturing trends.

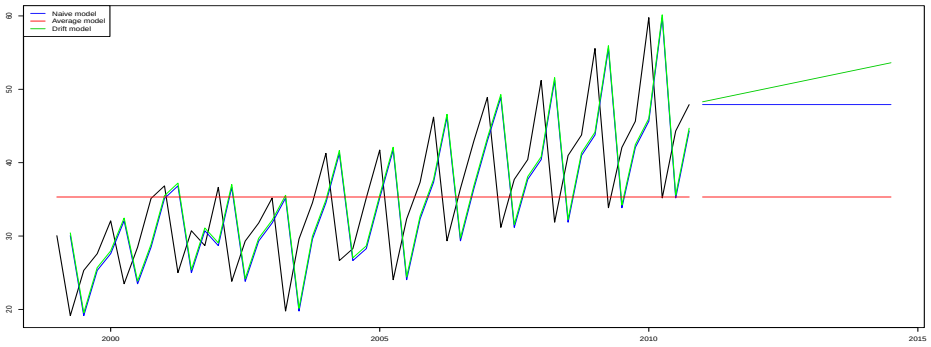

Figure 3.1: Simple models on *austourists*.

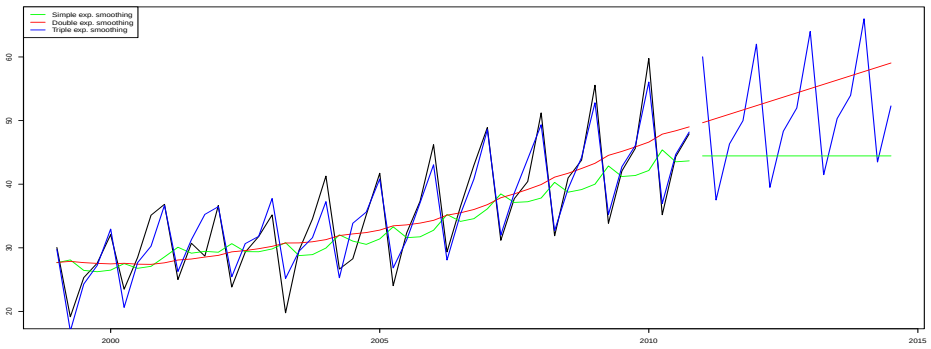

Figure 3.2: Exponential smoothings on *austourists*.

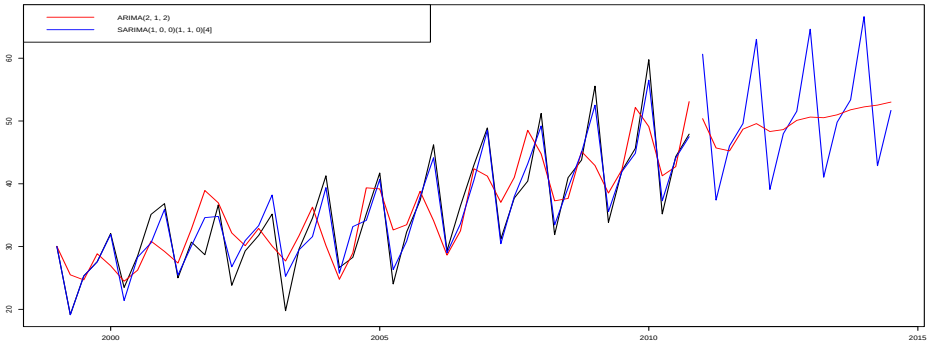

Figure 3.3: Arima models on *austourists*.

Exponential smoothings are depicted in 3.2. For this particular series it is obvious that seasonal is the best describing modelled time series. In the beginning can be seen how it 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 to quicier adapt to changes. Models with lower smoothing parameters are more robust to the changes.

Arima models are showed in 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.

In 3.1 are listed statistics produces by each model. Model with lowest one step ahead prediction is seasonal Arima and triple exponential smoothing which is correct for highly seasonal time series such as this one. For model selection would be used AIC statistics which is correlated to MSE.

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.

4 Existing Solutions

Before directly going to the design and implementation let's briefly look at existing monitoring solutions and libraries for time series modelling. In the next section are described not only direct competitors of Hawkular but also generally available software of this kind. Because Hawkular is still only an upstream project and not deliverable product for the customers thereby as alternative is considered Jboss Operations Network (JON). This work does not focus on comparison of monitoring applications. It is mentioned for more wider overview of this domain and also if some libraries can be used in the implementation.

4.1 Monitoring and Management Platforms

Out there are many monitoring solutions available. Some of them are offered as service (SaaS) and other need to be deployed in customer's environment. For each application are highlighted the most important features and how it differs from its rivals. It is also mentioned if the application offers predictive engine for metric data.

4.1.1 New Relic

New Relic was the first monitoring solution offered as SaaS [2]. It focuses on application performance rather than infrastructure view. Java applications monitoring is done by agent. Because agent manipulates with bytecote it is able to identify business transactions and monitor for instance utilization of REST endpoints Agent also automatically identifies databases connected to the application being monitored. From alert prediction perspective New Relic does not offer 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 instruments bytecode so the application is also able to monitor business transac-

tions. Ruxit differs from others that it offers root cause analysis rather than firing multiple alerts. It directly shows causing problems with visual representation [3]. Ruxit does not offer predictive alerting or charts.

4.1.3 Open Source Projects

As Hawkular is an open source project it is important to mention it's 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. By 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 these projects does 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. 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 is mentioned that predictive engine does not work on real time streams of data. Tivoli is definitely big solution and in order to get real predictive capabilities it would require to contact IBM support.

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`

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 smoothings. 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 error.

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 no any more maintained.

Performance of the library was also tested. Table 4.1 contains execution times of 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 `ets()` from `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 smoothing models would take about two and half hour. 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

2. Available at `http://www.stevengould.org/software/openforecast/index.shtml`
3. The most widely used: RCaller and JRI

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 allow native C++ invocation for optimized computations. Therefore package `forecast` can not be used with Rengin.

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

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

Table 5.1: Hawkular Data Mining modules.

In the following sections are described the most important parts of the implementation with respect to data structures and used algorithms.

5.1 Design of Data Structures

The core data structures from `datamining-forecast` package are interfaces for time series models 5.1and forecasters. This two interfaces are related but not the same. Each time series model should be capable of predicting and learning. The main difference between forecaster and time series model is that forecaster is designed for online

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

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();
    ...
     }
```

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 a web application, for instance classes for accessing data analysis objects of given metric.

The interface `SubscriptionManager` was designed for accessing time series analysis objects. Some of its methods are listed in 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 map where the key is tenant and value another map where the key is metric id and value finally 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);
    ...
```

```
}
```

`Owner` specifies the owner who enabled prediction. The core class for data analysis is `Subscription`. Currently only time series forecasting is available. This object holds an instance of an automatic forecaster. Class diagram of the most important classes from `forecast` and `api` is in A.2.

The build produces two web archives. The first one is designed for standalone usage and the second for deployment into Hawkular. Both uses forecasting capabilities from `api` and `forecaster`. REST api is implemented in `rest` and then reused in integration module `dist`. For REST api implementation the standard JAX-RS is used.

Tests for REST api are in separate artifact `itest` which allows reusing them, for instance in the main integration project.

Some of the service classes like `SubscriptionManager` are used in several other classes and also multiple implementation could exists (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.

5.2 Automatic Forecaster

The most important class in Data Mining for doing predictions 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 easy added.

The most importantly this class is capable of dealing with concept drift. It holds circular buffer of historical metrics and if statistical properties of 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. System currently supports two strategies. The first strategy periodically each n observations selects new model. The Second one is more sophisticated and it selects a new model only when the error of learning one step ahead predictions exceeds by x% error calculated on learning points when model was selected.

Model selection is based on information citerions from section 3.3. 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 is configurable.

5.3 Model Optimizers

Model optimizers are the most important part of the model implementation. If the parameters of model are not estimated correctly model would produce high forecasting error. The idea behind optimization is to find the best parameters of the model that one step ahead forecasting error is the lowest possible. In general number of steps can be more than one. Formaly

$$da \tag{5.1}$$

Then well defined objective function for minimization is passed to non – linear optimization algorithm from Apache Math Commons. This library implements several optimization algorithms: Nelder-Mead simplex, Virginia Torczon's multi-directional and Mike Powell's BOBYQA. These algorithms does not need computed derivatives of the cost function.

The best results were acquired from Nelder-Mead simplex algorithm. Criterions were optimization execution time and quality of the estimated parameters. Estimated parameters were compared to R's estimations.

Table 4.1 contains execution times comparison of `OpenForecast` and R implementation. This section addes execution time of Data Mining implementation. Complete execution times are listed in 5.2. Execution times of Data Mining's optimizers are in some cases faster than R.

5.4 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

Model	OpenForecast	R `ets()` from `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.

second web archive adds extra functionality for interacting with other Hawkular components.

Hawkular as a project is packaged as a modified Wildfly server with several Hawkular components deployed. In Hawkular all entities are stored in Inventory, therefore Data Mining uses this module for querying metrics definitions and it adds some extra attributes necessary for predictions. Another module which is used by Data Mining is Hawkular Metrics. It is a metric storage based on hybrid key-value database Apache Cassandra. Diagram 5.1 shows Data Mining integration into Hawkular.

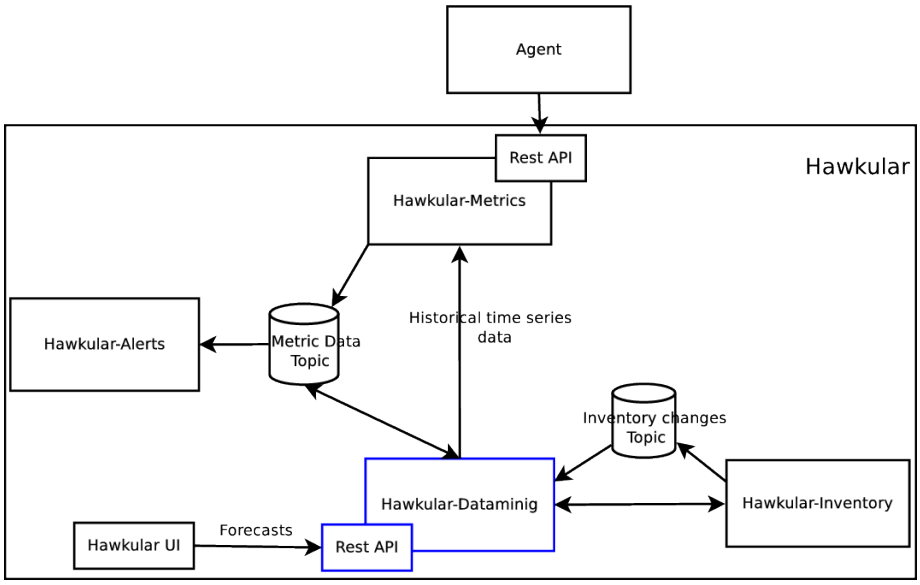

Figure 5.1: The integration into Hawkular.

When Data Mining is deployed into Hawkular users can enable predictions by creating Relationship[2] from tenant to the target entity. This 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 the figure 5.2 is depicted part of the structure of inventory with enabled predictions.

Relationships also stores properties needed for predictions. One of them is forecasting horizon which tells how many prediction steps are performed for each new incoming metric data points. With this approach of storing relationships in Inventory, Data Mining doesn't 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[3] is good approach for storing data model in this domain.

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

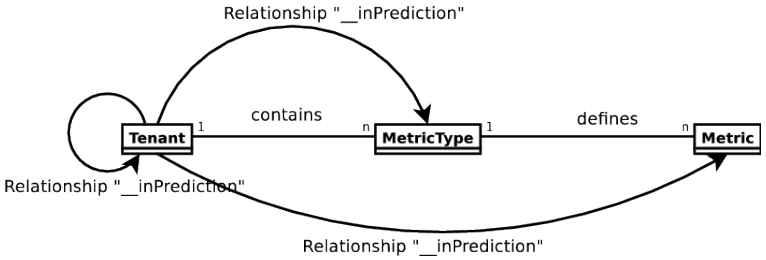

Figure 5.2: Relationship between entities of Inventory.

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 query and send it for execution to Inventory. As solution REST api 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.

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

Another two solutions were proposed. The first was to bypass Metrics and query data directly from Cassandra. The second was inject Metrics as CDI bean and use its Java api. The first solution was refused because the schema is maintained by Metrics and can eventually can change also the backed (Cassandra) can change. The final solution which is also implemented is injecting Metrics as CDI bean. This solution 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.

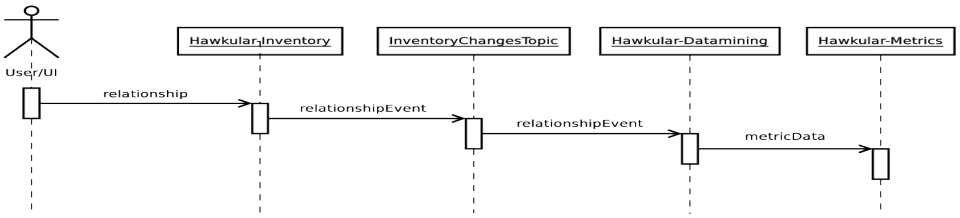

Figure 5.3: Sequence diagram of enabling predictions.

Diagram 5.4 shows complete data flow from agent to alerts evaluation engine. Metric data is sent by agent to Metrics REST endpoint and stored in 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 predicted data are sent to the same JMS topic and consumed by Alerts.

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. At this point an alert for predicted metrics can be triggered.

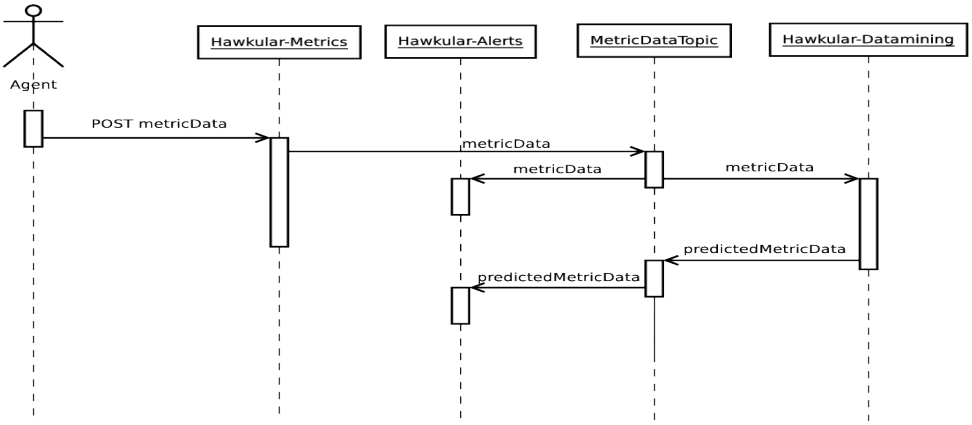

Figure 5.4: Sequence diagram of incoming data from agent.

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 run from Maven profile which starts web server with necessary modules deployed. Last type of implemented tests are end-to-end tests which calls REST endpoint of one module and expect result in call from another module. This type of tests are also executed in Maven profile with running web server. End-to-end tests for Data Mining project live in the main Hawkular integration repository[4].

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 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 [5].

5.6 Releases

By the time writing of 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 5.1. Release notes of the release are available on Github[6].

5.7 Retrospective

4. Available at `https://github.com/hawkular/hawkular`.
5. Available at `http://www.hawkular.org/docs/rest/rest-datamining.html`
6. Available at `https://github.com/hawkular/hawkular-datamining/releases`

6 Evaluation of Implemented Models

TODO Do an example how to generate an alert.

6.1 Prediction capabilities

Show prediction capabilities of implemented models. TODO Can I compare it against R implementation?

6.2 The Most Important Metrics

TODO select subset of the most important metrics and show on them predictions.

7 Conclusion

Bibliography

[1] Measuring time series characteristics. online.
URL http://robjhyndman.com/hyndsight/tscharacteristics/

[2] New Relic. 2016, [Online; accessed 20-April-2016].
URL https://en.wikipedia.org/wiki/New_Relic

[3] Root cause analysis of infrastructure issues. 2016, [Online; accessed 11-April-2016].
URL https://help.ruxit.com/display/RUXITDOC/Root+cause+analysis+of+infrastructure+issues

[4] Aras, S.; Kocakoç, İ. D.: A new model selection strategy in time series forecasting with artificial neural networks: IHTS. *Neurocomputing*, ročník 174, 2016: s. 974–987.

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

[6] COCIANU, C.-L.; GRIGORYAN, H.: An Artificial Neural Network for Data Forecasting Purposes. *Informatica Economica*, ročník 19, č. 2, 2015: s. 34–45, ISSN 14531305.

[7] Hyndman, R.; Athanasopoulos, G.: Forecasting: principles and practice. online.
URL https://www.otexts.org/book/fpp

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

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

[10] Wikipedia: Moving-average model. 2016, [Online; accessed 11-April-2016].
URL https://en.wikipedia.org/wiki/Moving-average_model

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

A Figures

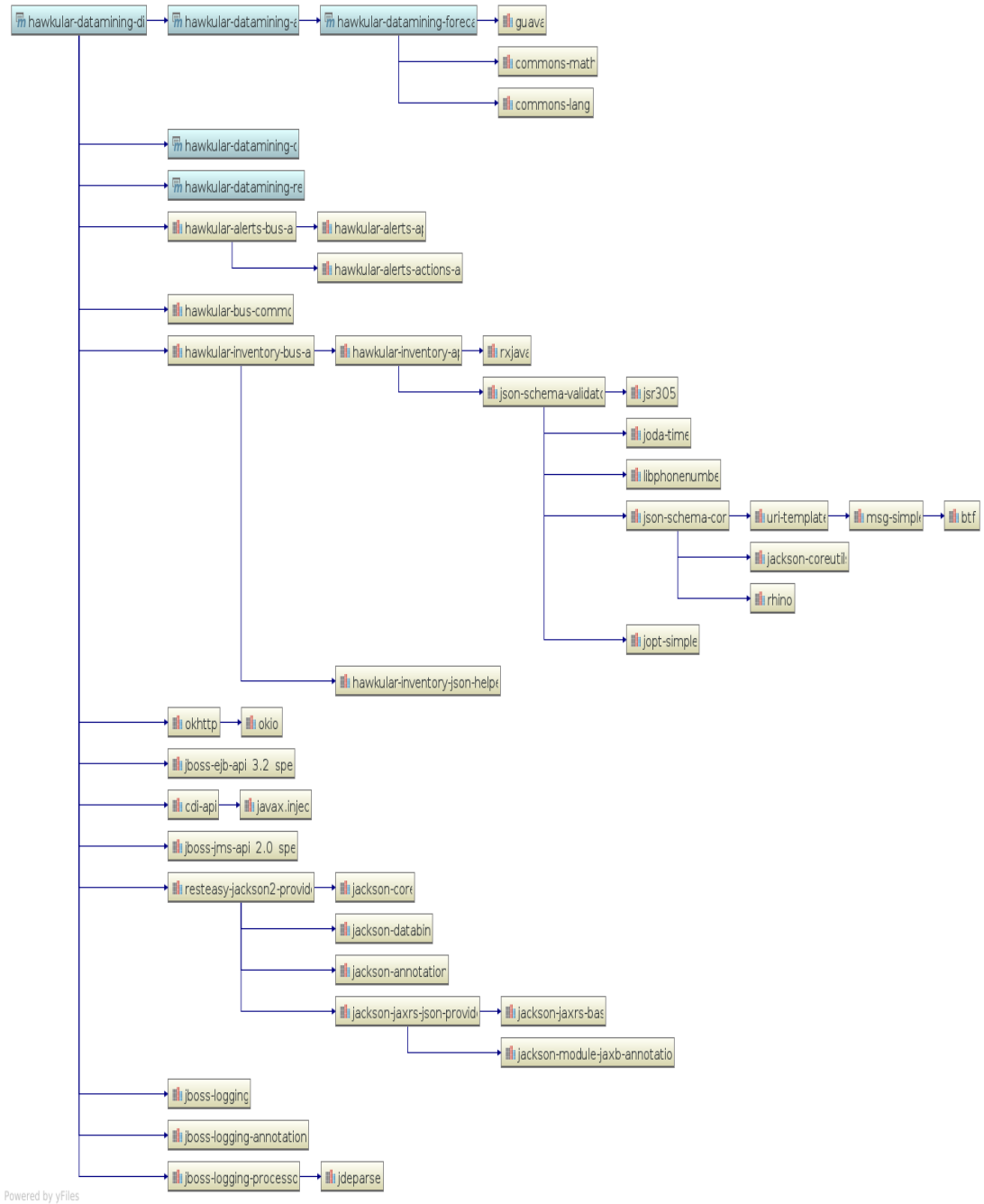

Figure A.1: Dependency tree of maven artifacts.

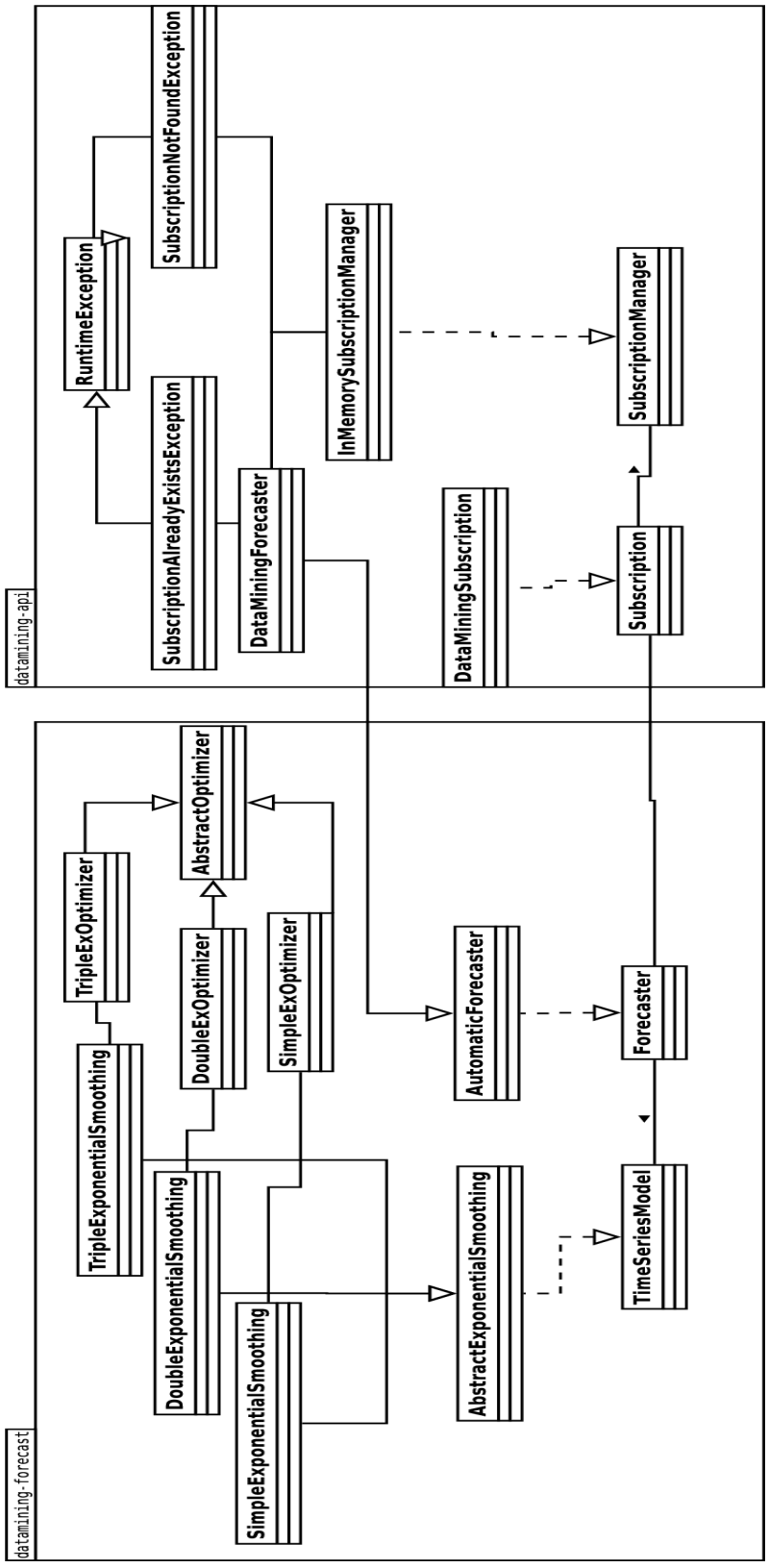

Figure A.2: Partial class diagram of `datamining-forecast` and `datamining-api`.