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

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. 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 Existing Solutions

TODO describe existing software.

3 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.

3.1 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.

$$\hat{y}_{T+h|T} = \overline{y} = (y_1 + \cdots + y_T)/T \tag{3.1}$$

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

$$\hat{y}_{T+h|T} = y_T \tag{3.2}$$

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

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T} y_t - y_{t-1} = y_T + h(\frac{y_T - y_1}{T-1}) \quad (3.3)$$

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.

3.2 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:

$$y = \beta_0 + \beta_1 x + \epsilon \quad (3.4)$$

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

$$SSE = \sum_{i=1}^{N} \epsilon_i^2 = \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i)^2 \quad (3.5)$$

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.

3.3 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 3.6.

$$\begin{aligned} \hat{y}_{T+1|T} &= l_t \\ l_t &= \alpha y_t + (1-\alpha) l_{t-1} \end{aligned} \tag{3.6}$$

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.

3.4 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.

$$\begin{aligned} \hat{y}_{t+h|t} &= l_t + hb_t \\ l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta(l_t - l_{t-1}) + (1-\beta) b_{t-1} \end{aligned} \tag{3.7}$$

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 $\alpha, \beta \in [0,1]$.

3.5 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 3.8. One for level, second for trend and third for seasonality. Each pattern uses smoothing constant $\alpha, \beta, \gamma \in [0,1]$.

$$\begin{aligned} \hat{y}_{t+h|t} &= l_t + hb_t + s_{t+h_m-m} \\ l_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \\ s_t &= \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m} \end{aligned} \tag{3.8}$$

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.

3.6 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 3.9. The order of this model is defined by p and can be determined from PACF function [7]. A moving average model (MA) is sum of weighted errors of order q. The order of this part can be determined from ACF function [7]. 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 [3]. For successful estimation a certain number of historical points needs to be available. In [7] minimal training size is set to at least fifty observations.

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t \qquad (3.9)$$
$$\epsilon_t \overset{iid}{\sim} N(0, \sigma^2)$$

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 [8]. 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 3.10.

$$y_t = \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_p y_{t-q} + \epsilon_t \qquad (3.10)$$
$$\epsilon_t \overset{iid}{\sim} N(0, \sigma^2)$$

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 [3]. ARIMA model is often written with backshift operator $By_t = y_{t-1}$. With this operator ARIMA(p, d, q) is listed in 3.11. On the left side of the equation is AR(P) process and on the right MA(q).

$$(1 - \phi_1 B - \cdots - \phi_p B^p)(1 - B)^d y_t = \qquad (3.11)$$
$$c + (1 + \theta_1 B + \cdots + \theta_q B^q)\epsilon_t$$

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

3.7 Artificial Neural Networks

Recently a large number of successful applications using neural networks for time series modeling show that they can produce valuable

results [9]. 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 [2]. 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 [4].

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.

3.8 Time series decomposition

In modelling time series it is sometimes necessary to decompose series to trend, seasonal and random component [5]. 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 3.12. Which one to use depends on the underlying time series model.

$$y_t = T_t + S_t + C_t + E_t \tag{3.12}$$
$$y_t = T_t \times S_t \times C_t \times E_t \tag{3.13}$$

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 substracted from each period.
- Random component is calculated by substracting trend and seasonal component from original time series $\hat{E}_t = y_t - \hat{T}_t - \hat{S}_t$.

3.9 Augmented Dickey – Fuller Test

Time series statistical tests are often used for testing if there is paricular 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 4.1.

$$\Delta y_t = \alpha + \beta t + \gamma \Delta y_{t-1} + \cdots + \delta p - 1 \Delta y_{t-p+1} + \epsilon_t \tag{3.14}$$

$$ADF = \frac{\hat{\gamma}}{SE(\hat{\gamma})} \tag{3.15}$$

There are multiple variants of ADF test. Some of them leave out some parts of equation 4.1. The most imporat 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 stationariry. 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 4.1. Then calculate ADF statistics with 3.15. SE denotes standard error of estimated $\hat{\gamma}$.

3.10 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 diffucult task and result often differs from correct period, specially if there is signaficant noise present in the series [6].

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 3.1 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.

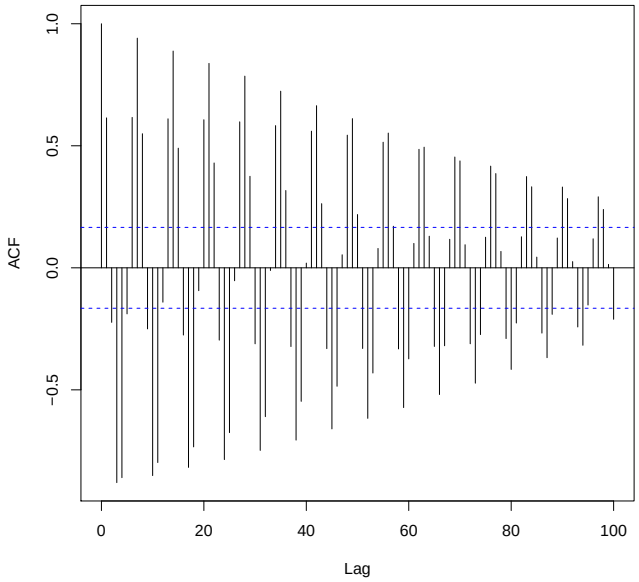

Figure 3.1: ACF of sine function.

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 singnificant 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
```

4 Models on Real Data

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 satistical system R.

4.1 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 monitonically decreasing or increasing and availability represents up or down state of a resource.

4.2 Evaluating Forecast Accuracy

In order to evaluate model it is important to estimate an error of the forecast. There are several methods for evaluating forecasting errors. Chosen were two MAE and RMSE. They are very similar however, RMSE gives relatively high weight to larger errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.1}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{4.2}$$

TODO show all charts of all described models.

5 Design and Implementation

Module for an alert prediction was named Hawkular Data mining. Source code[1] is versioned in Git hosted on Github. On every commit a build with integration and unit tests was triggered in Travis CI. Pull requests were always reviewed by some team member.

In the following chapters is described integration, design and the most important sections of implementation.

5.1 Integration with Hawkular

Data mining module had to follow architecture of the whole Hawkular application. The same approach was followed as in other modules. That means the module works also in standalone fashion without Hawkular. The build produces two Java web applications packaged as WAR. One is for standalone usage and other with integration code for Hawkular.

Integration with Hawkular is showed in 5.1. The module interacts with Inventory, Metrics and Alerts. User interface uses Data mining REST API for getting predictions for charts. Communication is done through Java messaging system and REST calls. Therefore modules are loosely coupled.

Metrics definitions and prediction metadata are stored in Inventory. Communication flows through JMS request response temporary queues. This was implemented specially for asking data across all tenants.

Forecasting of metrics in Hawkular is enabled by creating relationship from tenant to tenant, metric type or directly to metric in Inventory. Lower levels overrides higher (configuration on metric overrides configuration on metric type or tenant...). Every change in Inventory is sent to bus topic where other modules can consume it. When prediction gets enabled Data mining queries all historical metrics to initialize model.

When Metrics receives data from Agent it sends them to topic where is consumed by Alerts and Data mining. If metric if being fore-

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

casting model weight are updated and predicted values are sent to the same topic. Original and predicted time series are consumed by Alerts and conditions are evaluated. At this point an alert can be fired.

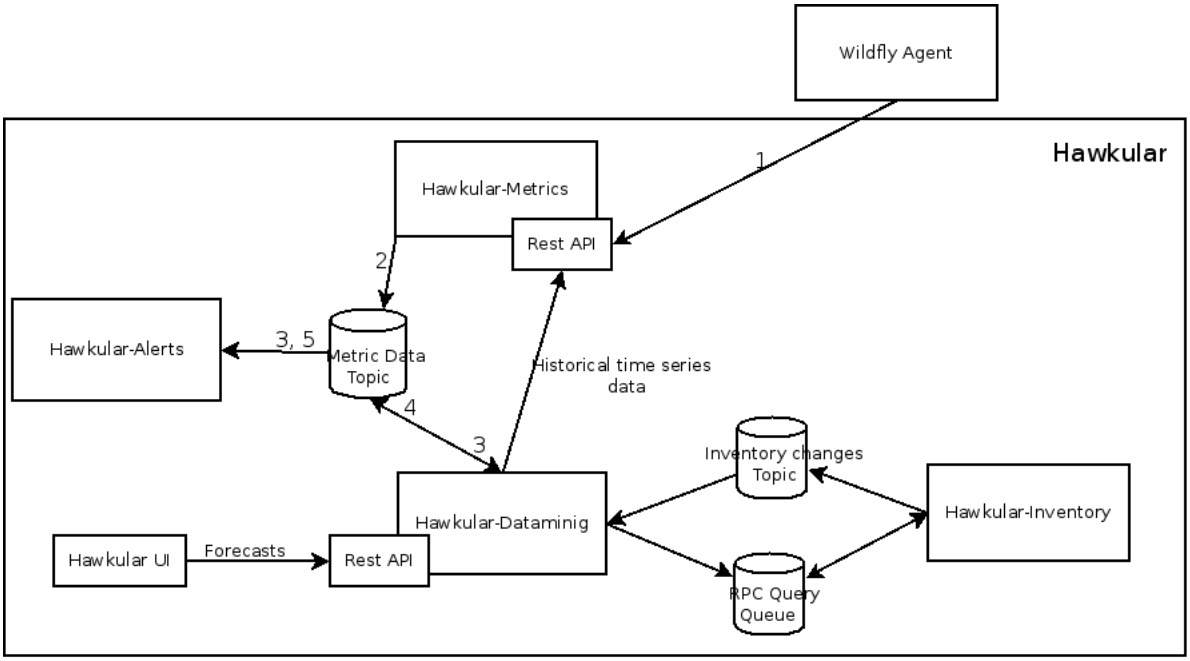

Figure 5.1: The integration with Hawkular.

5.2 Design of data structures

In this section are described the most interesting parts of the implementation.

Hawkular Inventory stores all entities of the application. Back end is graph database[2]. In the 5.2 is showed part of the entities which are important for Data mining module. Intentory high level API of-

2. Compatible with Apache Tinkerpop – Titan and TinkerGraph

fers creating relationships between arbitrary entities. This was used for enabling forecasting. This relationship contains properties map where is stored configuration of forecasting.

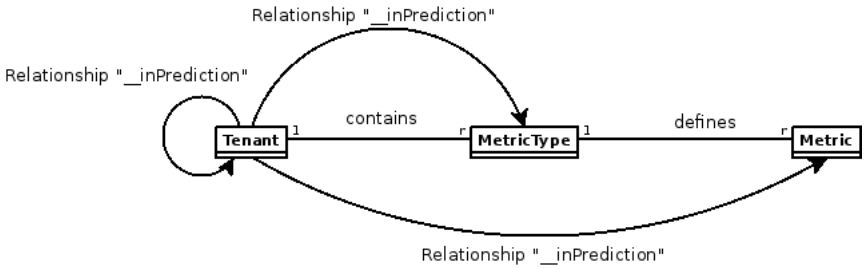

Figure 5.2: Structure of Inventory.

For subscribing predictions and holding models was designed interface `ModelManager` 5.1. Implementation of this interface holds in memory objects of models with respect to hierarchy of Inventory.

Listing 5.1: Interface Model Manager

```
public interface ModelManager {
    void subscribe(Metric, Set<ModelOwner>);
    void unSubscribe(tenantId, metricId);
    Model model(tenantId, metricId);
    ...
}
```

`ModelManager` is initialized on application startup and any change in Inventory is propagated through JMS to this object. With this approach Data mining is always synchronized with inventory.

Interface `ForecastingEngine` 5.2 provides forecasts for subscribed metrics. Implementation of this interface contains `ModelManager`.

Listing 5.2: Interface Forecasting Engine

```
public interface ForecastingEngine {
    void learn(List<DataPoint> ts);
    void List<DataPoint> predict(tenantId, metricId, nAhead);
    ...
}
```

5.3 Testing and Documentation

Unit tests were developed to cover crucial functionality of the program. The frameworks JUnit and TestNG are used for testing. Data mining module interacts with many other modules so integration and end to end tests were also implemented. Integration and end to end tests were written in Groovy because of the simple and well-arranged http client. It is also possible to easy define JSON string which is sent as POST object.

Documentation is written directly in Java code as javadoc. Documentation of the REST API was automatically generated by framework Swagger[3] and then automatically uploaded at Hawkular website. This was done at every build in Travis-CI. With this approach there were always the newest documentation available.

3. Available at <http://swagger.io/>

6 Evaluation

TODO do an example how to generate an alert. maybe generate synthetic data and use it as input for models.

6.1 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] 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.

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

[4] 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.

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

[6] 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.

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

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

[9] 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.