Data-driven methods in application to flood defence systems monitoring and analysis

Pyayt, A.L.

Citation for published version (APA):
Chapter 5. Artificial Intelligence Component

A short description of the anomaly detection approach as implemented within the artificial intelligence (AI) component is presented in this chapter.

5.1. Requirements for the early warning system component

According to the UrbanFlood project specification, each component of the early warning system (EWS) should have a number of common interfaces (Figure 5.1):

1. Data transfer (input and output): Java Message Service (JMS), web service (WS), file transfer (FTP/SCP) or network sharing (SMB, NFS).
2. Component configuration: XenStore (required for start-up), JMS, WS, file transfer (FTP/SCP) and network sharing (SMB, NFS) (required for runtime mode).
3. Component status notification: JMS, WS.
4. Debugging and administration (human-machine interface (HMI)): remote desktop and console (VNC, RDP, SSH, Telnet).

Figure 5.1. Component interfaces technology.

More details regarding the UrbanFlood EWS architecture can be found in [31] and [32].

5.2. Artificial intelligence component development

The artificial intelligence (AI) component was implemented in the Java programming language.

* Parts of this chapter were published in:


67
It uses all of the advantages of Java, e.g., the standard classes for XML, its support of network computing, and its cross-platform features.

The core of the AI component is the Siemens Machine Learning (SML) library [110]. SML is a cross-platform software toolbox that contains numerous algorithms for signal processing and machine learning. SML is primarily implemented in the C++ programming language. For this reason, the Simplified Wrapper and Interface Generator (SWIG) tool was used to generate Java Native Interface (JNI) code for connection of blocks written in C++ with blocks written in Java.

The last specification of the AI component interfaces is presented in Figure 5.2:

1. JMS was selected as one of the data transferring buses. The input data contain measurements from the dike in XML format. The results of data processing from different components are published to the output data stream. The output message of the AI component is also prepared in XML format and contains the calculated confidence value of levee normal behaviour along with timestamps of the measurements and analysis. The input measurements and calculated abnormal confidence values are visualised using the WebDashBoard internal component (Figure 5.3). An example of visualisation is shown in Figure 5.7.

2. Configuration of the EWS component is performed by the XEN store, which provides the network parameters required for the initialisation of the AI component.

3. Each component should provide status messages to fulfil the EWS self-monitoring task. Thus a separate thread of the AI application sends messages with component status in XML format to the JMS bus at the specified rate.

4. The HMI is provided by the administrator of the virtual machine on which the component is deployed. VNC and SSH, for example, are actively used for testing and debugging purposes on the remote machine.

There are two classes of input data: online input measurements in the testing mode and large dump files with historical measurements (the size depends on the selected time interval of the data) in the training mode. Both of them are in XML format. There are multiple different XML parsing technologies available: the Document Object Model (DOM), Simple API for XML (SAX), Streaming API for XML (StAX), and Java Architecture for XML Binding (JAXB).
JAXB provides a fast and convenient method to bind XML schemas and Java representations, thereby making it easy for Java developers to incorporate XML data and processing functions in Java applications. JAXB also provides a method to generate XML schema from Java objects [17], so-called marshalling/unmarshalling. The unmarshalling process optionally involves validation of the source XML documents. Validation is the process of verifying that an XML document meets all of the constraints expressed in the schema.

StAX has a set of classes to write XML documents, which SAX, for example, does not treat at all. Unlike DOM or any other tree-based parser, the document does not remain in memory while it is being built [79]. That is why StAX technology provides high-performance stream filtering, processing, and modification, particularly with low memory and limited extensibility requirements. Detailed specifications can be found in [17].

In dike monitoring, there is a large amount of sensors and dumps in the training mode, and the performance of the XML parsing is a critical issue. Thus, StAX technology is preferable for reading input messages and publishing output results.

5.3. **AI component of the UrbanFlood EWS**

The AI component of the UrbanFlood EWS is presented in Figure 5.3. AnySense is a database that collects all of the measurements from the sensors gathered in the sensor cabinet. These data are distributed among all components of the EWS through a Java Message Service (JMS) bus in XML format and consumed by the AI component. The data analysis block consists of various data-driven methods for implementing data pre-processing, feature extraction and anomaly detection. The results of the data analysis, which contain confidence values of dike normal behaviour, are sent to the JMS bus to be received by the Decision Support System (DSS) and other components [99].

![Figure 5.3. Artificial intelligence (AI) component of the UrbanFlood early warning system. The AnySense data base stores on-line measurements.](image-url)
Multiple one-sided classifier instances can be created within one AI component, and several AI components can be created within one EWS for monitoring. This component architecture efficiently utilizes the cloud-computing infrastructure of the EWS. Each AI component instance is a separate virtual machine (VM) that runs on a virtualization host. Any required number of AI components can be started and configured [69]. An example visualisation of the results of the AI component is presented in Figure 5.4.

Figure 5.4. (a) Screenshot of the multi-touch table with the Artificial Intelligence (AI) component output and Livedike cross-sections. (b) Visualization of confidence values calculated by the AI component for the selected sensor from the selected cross-section.

### 5.4. AnySense Messages Generator

The AnySense Messages Generator (ASMG) reads XML dumps of historical measurements for all sensors and sends one message per sensor to the JMS topic at a rate specified by the user (e.g., the AnySense messages with the sensor measurements). In this case, the ASMG works as an “artificial” dike and is able to simulate abnormal dike behaviour upon the user submitting a request through a web interface (Figure 5.5 and Figure 5.6). ASMG was developed as an independent EWS component and can be easily used for testing purposes for other components.

Figure 5.5. Interfaces of the AnySense messages generator (ASMG) component.
5.5 Combination of AI and ASMG components

The ASMG component generates the dike measurements that are related to normal behaviour and sends them to the JMS bus. The AI component reads and analyses these data. For each sensor with at least two parameters, one neural cloud is trained. Then, an anomaly is generated upon the user’s request (timestamp 36). This anomaly is successfully detected by NC, as indicated by the blue line between timestamps 36 and 37. Then, the abnormal regime was deactivated (timestamp 39) and the NC exhibits normal behaviour (Figure 5.7).

Figure 5.6. AnySense messages generator (ASMG) component as part of the UrbanFlood early warning system.

Figure 5.7. (a) Combination of the Artificial Intelligence (AI) and AnySense messages generator (ASMG) components. (b) Visualisation of artificial anomaly detection. The dark blue line confidence values of normal behaviour (close to 1 indicates normal behaviour, whereas values close to 0 indicate anomalies); the brown and light blue colours represent measurements.
5.6. Evolution of the AI component

This section presents the stages of the AI component evolution. In the first stage only one set of the one-side classifiers was used (Figure 5.8). The first results for the Zeeland dike and Stammer dike data analysis are related to this period. A special component for downloading the historical measurements from the AnySense database was prepared. A manual download procedure was required for preparation of the training data set. Manual changes to the Java source code were required for updating the training procedure inside the AI component (e.g., to change a set of neural clouds or the set of sensors to be used as the training set); see Figure 5.9.

![Figure 5.8. First version of the AI component workflow.](image)

The second generation of the AI component is presented in Figure 5.10. A feature extraction procedure was added. This required updating of the AI component training procedure (Figure 5.11). The training set still had to be downloaded manually from the AnySense database. A configuration file that describes two stages of data processing (feature extraction and classification) was prepared but creation and modification of the overall data analysis workflow still required manual modifications of the Java source code.

![Figure 5.9. First version of the Artificial Intelligence (AI) component training method.](image)
Figure 5.10. Second version of the Artificial Intelligence (AI) component workflow.

The third generation of the AI component (called flexible AI, or Flex AI) is presented in Figure 5.12. All required stages of data analysis were implemented. Manual tuning of the data processing workflows became too complicated; thus, a configuration file was used for management of the overall data processing chain (Figure 5.13).

Figure 5.12. Third version of the Artificial Intelligence (AI) component workflow.
The operator of the system had to work with only the XML file; all other procedures (e.g., downloading historical measurements and training of the one-sided classifiers) were performed automatically. Development of this tool required involvement of several experts from Siemens LLC. Development was performed using the Spring framework [19].

The experts also implemented a configuration tool for creation of the XML configuration files in a visual manner. A library of visual elements, e.g., neural clouds, was developed. Data analysis blocks are connected in the workflow (Figure 5.14). The Eclipse Modelling Framework [5] was used for this task.

Figure 5.13. Third version of the Artificial Intelligence (AI) component training method.

Figure 5.14. Screenshot of the configuration tool.
5.7 Conclusions

This chapter is dedicated to development of the anomaly detection approach as a component of the early warning system (*fourth scientific objective*): a short overview of the development of the artificial intelligence (AI) component is presented.

The selected one-side classification based anomaly detection concept was implemented within this component. Increasing complexity of data processing required more flexibility of the AI component for generation of cascades on data processing. Evolution of the AI component is presented in this chapter. The last version of the AI component (Flex AI) provided flexible management of the data analysis workflow: only one XML configuration file had to be created in order to construct the full data analysis chain. A visual tool for preparation of XML configuration file was developed by the experts.

The development of the additional component (the AnySense Messages Generator) is also presented in this chapter. This component was used for testing purposes.

There are several well-known data analysis tools like Weka data mining software [21] or Knime [9] that could be theoretically adapted to the tasks of the project. There were several reasons why this was not done and why the new component was developed.

The first reason was the natural process of the component development. The first version of the AI component had to be prepared on a short timescale, which is why development in pure Java was selected after considering the requirements for interfacing with the component described in Section 5.1. Filling of the component with data driven methods was a continuous process because the complexity of the anomaly detection task could not be evaluated in advance.

The second reason was the requirement of on-line processing. The Knime software package, for example, provides only a single calculation of input data. If the regular automatic execution is required, than the server license has to be obtained.