FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Trading anomaly detection framework
Student:	Josef Havelka
Supervisor:	Ing. Jan Blizničenko
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2021/2022

Instructions

The goal of the thesis is to create a framework for stock-exchange trading anomaly detection. The framework must allow its user to create and use algorithms that can be integrated into existing business monitoring services used by the customer without further modifications.

Instructions:
- Analyze customer requirements and services used by the customer
- Research frameworks with a similar focus
- Research relevant technologies, tools and libraries
- Design the resulting framework
- Implement the software using Java programming language
- Demonstrate the functionality of the framework by using it to create an algorithm that works with data supplied by the customer
- Test and document your solution

Electronically approved by Ing. Michal Valenta, Ph.D. on 22 October 2020 in Prague.

Bachelor thesis

TRADING ANOMALY DETECTION FRAMEWORK

Josef Havelka

Czech Technical University in Prague, Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Jan Blizničenko
May 12, 2021

Link to the thesis: Josef Havelka. *Trading anomaly detection framework.* Bachelor thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Contents

List of Figures

List of Tables

List of code snippets

I want to thank my supervisor Ing. Jan Blizničenko, for helping me with the thesis. He was very responsive and was giving a lot of practical advice. I want to thank Bc. Aleksandr Karpenko for an idea that led to the assignment. Aleksandr was guiding me through the thesis most of the time. I want to thank Bc. Maria Karzhenkova for reviewing my work and consultations. I want to thank my schoolmate Patrik Vodila for his recommendations on the machine learning field. Finally, I want to thank the whole Rapid Addition company for the opportunity to combine school and work task in this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 13, 2021 .....

Abstract

This bachelor thesis aims to implement software helping to deploy an anomaly detection model into a Java program. Requirements of a customer that assigned this task were taken into account when developing the software. Several model formats were researched for storing a completed anomaly detection model. Also, relevant libraries for producing and consuming such models were studied. The knowledge of model formats and a basic understanding of an anomaly detection workflow was transformed into the resulting software written in Java that can be extended to use new formats. It can be integrated into any project running on Java programming language. Functionality is tested on data provided by the customer. The solution is intended to be filled in a future with algorithms analyzing data from the customer's scope of interest. The final software is published under an open-source license.

Keywords Java, anomaly detection framework, anomaly detection, machine learning model file formats, PMML, TensorFlow, ONNX, PFA

Abstrakt

Tato bakalářská práce si klade za cíl implementovat software pomáhající nasadit model detekce anomálií do programu napsaného v jazyce Java. Při vývoji softwaru byly brány v úvahu požadavky zákazníka, který zadal tento úkol. Pro uložení hotového modelu detekce anomálií bylo prozkoumáno několik formátů, do kterých se model ukládá. Byly také studovány relevantní knihovny pro výrobu a spotřebu takových modelů. Znalost formátů modelu a základní znalost pracovního postupu detekce anomálií byla transformována do výsledného softwaru napsaného v Javě, který lze rozšířit tak, aby používal nové formáty. Může být integrován do jakéhokoli projektu běžícího v programovacím jazyce Java. Funkčnost je testována na datech poskytnutých zákazníkem. Řešení má být v budoucnu naplněno algoritmy analyzujícími data z domény zákazníka. Konečný software je publikován pod licencí open-source.
Klíčová slova Java, framework na detekci anomálií, detekce anomálií, formáty ma ukládání modelů strojového učení, PMML, TensorFlow, ONNX, PFA

Summary

Motivation

Nowadays, machine learning used for predictions or anomaly detection goes rapidly forward. Nothing is perfect, so it is priceless to get notified about an undesired behaviour by artificial intelligence.

The most popular for machine learning are Python, R or C++ libraries. The machine learning model is saved into a file in some format. Creating a model and processing the data needed for training a model is more convenient and more straightforward in these libraries. On the other hand, several applications run on a different programming language other than the three mentioned above. There is a need of separating the model creation and model deployment.

Fortunately, there exist approaches to how to use a prepared machine learning model in an application. The first way is to provide the model scoring as a service application scoring the model in the language in which it was originally created. The second is to transfer the model stored in a file and evaluate the model right in the application. The machine learning community has developed several formats in which a model can be transferred. Each file structure of a transported model has its pros and cons. A brand new format for storing a model can come any day now.

Goal

The main goal is to implement an anomaly detection framework, which will load anomaly detection models stored in some file format and use them on the incoming data in real-time. A user will provide the models. The framework will be written purely in the Java programming language. The customer will be able to use the resulting framework without further modifications inside the customer's already existing Java project.

The research part aims to get familiar with frameworks with a similar focus and relevant technologies, tools, and libraries. Namely, it is important to research commonly used formats for storing models, machine learning libraries for the export/import of this model into/from the corresponding file format. Last but not least, this requires some basic knowledge of anomaly detection algorithms.

The practical part aims to design the resulting framework, implement the software using Java programming language, and demonstrate its functionality. The functionality will be tested on the customer's data. Finally, the whole solution will be documented.

Steps

Firstly it is essential to get familiar with the basics of machine learning. Then comes collecting and analyzing customer requirements to clarify details of what needs to be implemented.

The next step is to study frameworks and projects with a similar focus to get inspired.

Afterwards, there is a need for the author of this thesis to research relevant tools, technologies and libraries. The most important bit is to focus on options of evaluating models directly in Java run-time.

Knowledge from previous stages is transformed into the resulting framework covered with a reasonable amount of unit tests. A user can extend the framework with new model formats.

The functionality of the framework is demonstrated on examples, including examples with data provided by a customer.

The very last stage consists of documentation and description of the resulting software.

Results of the thesis

The main product of the thesis is the implemented extensible framework with examples of usage of a PMML and TensorFlow's custom format. Also, there is relatively detailed research about recently used formats for storing a model and corresponding libraries for creating models and running them in a Java run-time.

Conclusion

In some way, this thesis is a continuation of Martina Fusková – Anomaly Detection for Stock Market Trading Data and Aleksandr Karpenko – Design of anomaly detection for stock market trading, both researching and optimizing anomaly detection algorithms suitable for detecting outliers in the electronic trading domain.

The upcoming text is more about studying and applying technology for deploying and operating an anomaly detection model than searching for the best machine learning algorithms and methods. There are some signs that the company, who set the task of implementing such a framework, will soon use the final framework. The project is under an open-source license, so everybody can go through it or extend it.

List of abbreviations

FIX	Financial Information eXchange
EGADS	Extensible Generic Anomaly Detection System
JSON	JavaScript Object Notation
YMS	Yahoo Monitoring Service
ADM	Anomaly Detection Module
DM	Deviation Metric
AM	Alerting Module
PMML	Predictive Model Markup Language
ORE	Oracle R Enterprise
OSX	Oracle Streaming Explorer
DAEEPaaS	DEEP as a Service
API	Application Programming Interface
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
SVM	Support Vector Machine
DMG	Data Mining Group
XML	Extensible Markup Language
PFA	Portable Format for Analytics
ONNX	Open Neural Network Exchange
JVM	Java Virtual Machine
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPGA	Field-programmable Gate Array
UML	The Unified Modeling Language
PLU	Price Look-Up Code
LSTM	Long Short-term Memory
S&P 500	Standard and Poor's 500
RNN	Recurrent Neural Network
API	Application Programming Interface

Chapter 1

Theoretical background

This chapter will introduce the terminology and theoretical background needed for an understanding of the following document.

1.1 Introduction to anomaly detection

"*Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior. These nonconforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants in different application domains.*" [1]

"*Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment.*" [2]

In this thesis, we will focus on using machine learning algorithms for anomaly detection. According to the survey [1], the input for the anomaly detection is a set of data instances – also referred to as points, objects, vectors, observations, patterns, etc. Each data instance holds a collection of attributes – also referred to as variable, characteristic, feature, field, or dimension.

When taking the input described above for outlier detection, there are three kinds of anomalies [1]:

Point Anomalies – A single data instance is considered anomalous concerning the rest of the data.

Contextual Anomalies – A single data instance is considered abnormal in some specific context but not otherwise.

Collective Anomalies – A group of data instances is considered anomalous, considering the whole data set.

As described in [3], there exist three methods of model learning:

Supervised – The training data set contains labelled data instances. For example, tagging what is an outlier and what is not.

Semi-supervised – Only a part of the data is labelled.

Unsupervised – The data is unlabelled. This technique relies on the fact that the normal instances make the majority of the data set.

1.1.1 General machine learning workflow

The article [4] confirms the general machine learning workflow has three stages:

1. Processing of the input data
2. Learning or training of the underlying model with training data
3. Scoring of the machine learning model to predict or make decisions on new data

1.1.2 Anomaly detection workflow

The workflow consists of three main parts of anomaly detection. The first one is data preprocessing. It means, for example, selecting important features or scaling the data. The second part is to apply some prediction or decision-making model to produce forecasts or decisions. The output of the second step is then post-processed. Post-processing can mean filtering out the product of a previous stage with some rules and then labelling a subset of the original input data as anomalies.

1.2 Electronic trading

This section introduces the basics of electronic trading for a better understanding of the customer's business domain.

1.2.1 Instrument

"*In the field of finance, an instrument is a tradable asset, or negotiable item, such as a security, commodity, derivative, or index, or any item that underlies a derivative.*" [5]

1.2.2 Investor/Client

An investor can be an individual or a firm.

1.2.3 Broker

"*A broker is an individual or firm that acts as an intermediary between an investor and a securities exchange.*" [6] As further described on Investopedia [6], the purpose of a broker is to make trading more convenient for an investor because everybody who wants to send orders to an exchange needs to be its member. The membership requires some responsibilities from the member's side. A broker is a member of an exchange and provides access to it as a service to the broker's clients.

1.2.4 Stock Exchange

A stock exchange is a place where the actual trading of securities takes place.

1.2.5 Financial Information eXchange (FIX) Protocol

As stated on Investopedia [7], FIX became de-facto a standard of electronic trading communication. The protocol contains a specification for each message that can be sent. The most used messages are:

New Order Single – a message representing that client wants to buy/sell an instrument

Order Cancel Replace Request – a message modifying a sent order by a client

Order Cancel Request – a message cancelling a sent order by a client

Execution Report – a response to the client reporting a state of an order placed on an exchange

The complete documentation of the protocol can be found on the FIXimate page [8].

Chapter 2

Customer requirements

2.1 Customer domain

The customer's scope of interest is electronic trading technology for financial markets. The customer has been developing software for routing and monitoring FIX messages in between brokers and exchanges.

2.1.1 Customer's motivation

Data analytics and machine learning engineers are using several libraries for training a model. Most of them are Python, R, C++, and Java libraries. The training can take much time, so the trained model is stored into a file. The framework's primary purpose is to separate the training of the model from the actual deployment and usage of the model. The customer wants to use the model purely in the Java code, but on the other hand, he does not want to make creators/trainers of the model dependant only on the Java libraries. They are free to use any library they like, and a user of the framework will provide the conversion from the file containing the trained model into the Java code.

A user will provide a model in two ways, a model stored in a file in some format or implemented directly in the Java programming language. For the first case, the user has to implement the conversion from the file format into the framework's representation of the anomaly detection model. The conversion means to parse the data from the file and implement the model in the Java programming language. In practice, it means to use some Java machine learning library or make the full implementation of the algorithm from scratch.

2.1.2 Customer's used technology

The Java backend platform handles the business logic of routing and monitoring the FIX messages. The customer calls the connection with a trading entity a session. Customer's monitoring service extracts and holds relevant features about the trading. The monitoring service is called Summary View. There are kept many calculated features, for example:

Quantity Sell – sum of the quantity of sold stocks

Messages – amount of messages sent

Orders – amount of new order singles sent

The routing logic can be adjusted and viewed in the C# frontend, taking the data directly from the backend.

2.1.3 Planned usage

The framework's intended usage is to add it as a module right into the backend system. The anomaly detection model will then be integrated and used directly inside the system with the framework's help. There are several places where it makes sense to deploy the anomaly detection model. Therefore the model has to be flexible. Examples for where some anomaly detection model can be applied are:

Summary view – checking anomalies on the subset of features from the summary overview

Session's latency – checking the time between sending a message and receiving a response to the message.

Session's throughput – checking a number of messages sent through the session per time period.

Concrete scenarios should be discussed with an expert on trading.

2.1.4 Requirements

The concrete requirements are described in this section. Most of them can be inferred from the previous text. After consultation with the customer, we have agreed on the following requirements.

2.1.4.1 Functional

Model loading – Implement the loading of a model from a file.

Model storing – Implement the storing of a model into a file.

Model feeding – Implement the real-time feeding of a model with data.

2.1.4.2 Nonfunctional

Extensibility – It has to be easy to extend the functionality of the framework.

Model formats – A user should be able to implement support for new model file formats.

Language – The framework has to be written purely in a Java programming language.

Integration – The customer has to be able to integrate the framework as a module into his already existing Java project without further modification.

Time efficiency – The solution has to be time-efficient. Alternatively, easy to optimize.

Chapter 3

Projects with a similar focus

Frameworks and projects with a similar focus as my thesis has will be introduced in this chapter. There are many public libraries for machine learning, but this section is not about these libraries. The main focus will be on frameworks helping with an already-trained model into production.

There is a limited palette of supported model formats for each project mentioned below. They usually choose a few supported formats and instead focus on another functionality. The thesis framework is also unique because models will be used directly inside the Java project. The very favoured solution is using a Client-Server architecture. The main pros of this architecture are language independence and easier deployment for new models. There is no need to reimplement an algorithm into the desired language.

On the other hand, the con is time efficiency. Direct implementation of the model inside the Java project eliminates the delay caused by the Client-Server communication. There exist libraries for parsing and evaluating the model from some formats directly in the Java run-time. These libraries will be described in the next chapter.

3.1 EGADS

"*EGADS (Extensible Generic Anomaly Detection System) is an open-source Java package to automatically detect anomalies in large scale time-series data. EGADS is meant to be a library that contains a number of anomaly detection techniques applicable to many use-cases in a single package with the only dependency being Java.*" [9]

The framework works only with time-series models. As mentioned in the presentation attached to [9], the framework supports only unsupervisely trained algorithms. This framework focuses mainly on advanced methods for recognizing several types of anomalies. The table 3.1 is a list of time-series features that are calculated for identifying a specific kind of an outlier.

3.1.1 Model formats

The author of the EGADS framework does not describe the model's storing and loading in much detail. As apparent from the source code fragment 3.1, trained anomaly detection models are saved using the JSON [10] format or serialized into a file. The model has to be trained using the framework strictly. This training part makes the framework a bit clumsy to use.

Table 3.1 Time-series features [9]

Time-series feature	Description
Periodicity (frequency)	Periodicity is very important for determining the seasonality. Trend Exists if there is a long-term change in the mean level
Seasonality	Exists when a time series is influenced by seasonal factors, such as month of the year or day of the week
Auto-correlation	Represents long-range dependence.
Non-linearity	a non-linear time-series contains complex dynamics that are usually not represented by linear models.
Skewness	Measures symmetry, or more precisely, the lack of symmetry.
Kurtosis	Measures if the data are peaked or flat, relative to a normal distribution.
Hurst	a measure of long-term memory of time series.
Lyapunov Exponent	a measure of the rate of divergence of nearby trajectories.

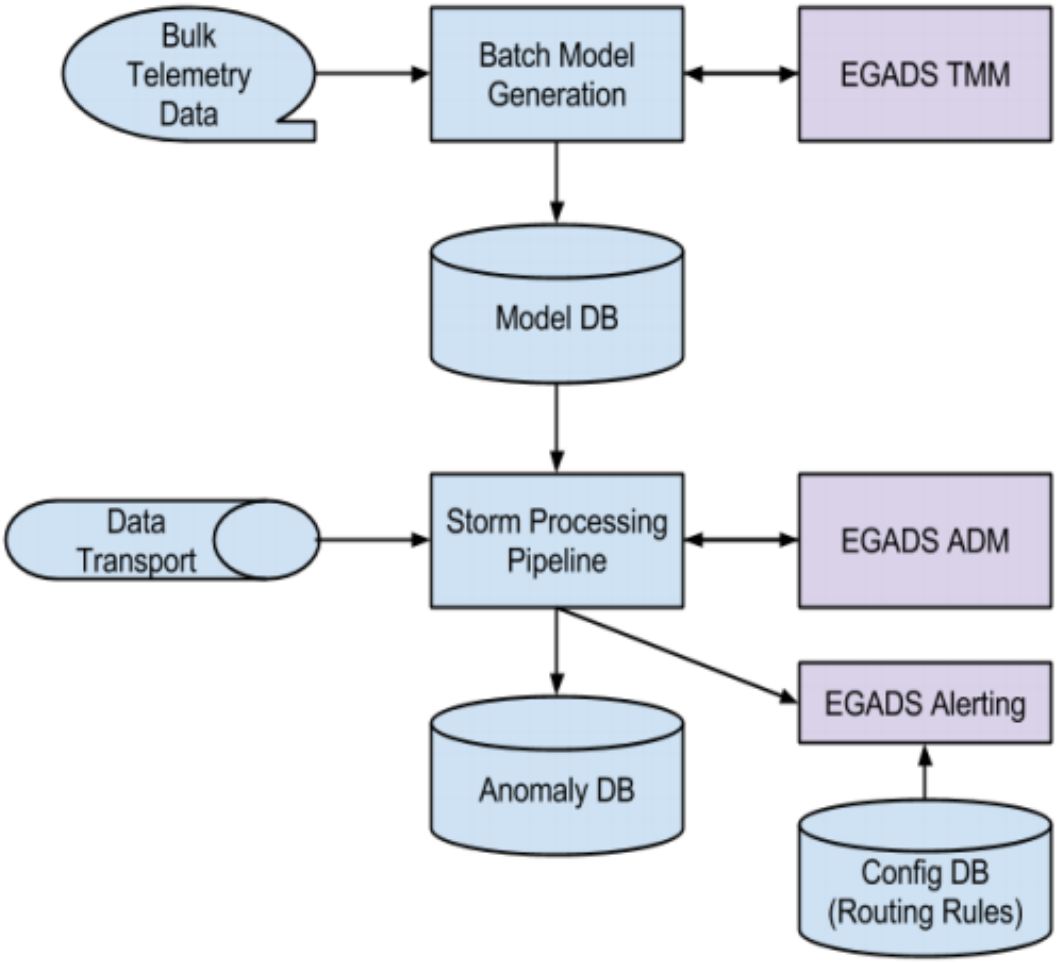
Code snippet 3.1 EGADS Model Interface [9]

```
package com.yahoo.egads.data;

import java.io.Serializable;

public interface Model extends JsonAble, Serializable {
...
}
```

Figure 3.1 EGADS-YMS architecture [9]



3.1.2 Workflow

Interesting is to look at the actual usage of the framework. "*EGADS operates as a stand-alone platform that can be used as a library in larger systems.*" [9] The actual integration with the Yahoo monitoring service (YMS) is viewed in the picture 3.1. The parts of the EGADS project are purple-coloured. Here it is used as a library for processing batches of input data.

In the beginning, the time-series modelling module (TMM) predicts a value of the datapoint. "*Given a time-series* $X = \{x^t \in R : \forall t \geq 0\}$*, the TMM provides the predicted value of* x^t *at time* t*, denoted by* u^t*.*" [9] Then the anomaly detection module is used (ADM). "*Given the predicted value* u^t *and the actual observed value* x^t*, the ADM computes some notion of deviation which we refer to as the deviation metric (DM).*" [9] "*These error metrics, together with other features, such as the time series characteristics, are used in the alerting module (AM), described in Section 4, to learn consumer's preferences and filter unimportant anomalies.*" [9]

3.1.3 Summary

The similarity of the EGADS and the thesis framework is that it is possible to implement and add new anomaly detection models, and both projects can be used as a Java library. However, the EGADS is more focused on advanced techniques of anomaly detection itself. It does not support adding new model's storing formats.

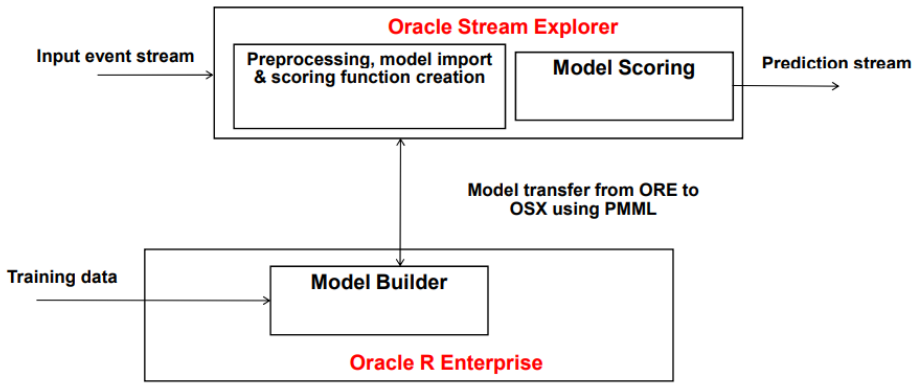
3.2 Oracle

Information about the architecture described in the section 3.2.1 was taken mainly from [11], which is from the year 2016. The section 3.2.2 taking knowledge from [12] will represent a different more up to date project from the year 2019.

3.2.1 Oracle streaming

Oracle has created a multi-functional, complex system for training a model on data from a database, converting it into the PMML format, and using it for predictions on the real-time data stream. Detailed information can be found in the presentation [11]. The complete architecture is captured in picture number 3.2.

Figure 3.2 Oracle Streaming architecture



[11]

The Oracle R Enterprise (ORE) uses R language scripts to create a model from the database's data. ORE exports these models into the PMML format and passes them for consumption to the Oracle Streaming Explorer (OSX). OSX takes the PMML file with a stored model. OSX loads it with the JPMML library, evaluates the incoming data from a real-time stream using the JPMML evaluator, and produces an output prediction stream.

3.2.2 Oracle machine learning

As apparent from a more recent paper [12], Oracle came up also with another approach. Oracle started using commonly used python and R machine learning libraries wrapped into the Oracle interface directly in a database and storing a result of an anomaly detection into a database. The anomaly detection result is then read from the database by the application, which needs the data with labelled anomalies.

3.2.3 Summary

An interesting discussion would be why it was not sufficient for Oracle to use only the PMML format to store models. As described in the next chapter, only some machine learning libraries support saving a model into this format. The next bottle kneck is the JPMML library described in the next chapter. It has a limited range of implemented anomaly detection algorithms. I assume that Oracle has use-cases when the higher latency with writing into and reading from a database does not matter. And for those use-cases, it is beneficial to dispose of a broader machine learning libraries arsenal.

3.3 DEEPaaS

"*DEEPaaS API is a Python package that provides a REST API (Fielding, 2000) that can be used to easily expose the underlying model functionality over HTTP.*" [13] As described in [14], a user can access a python model (implemented with a library of the user's choice) via a REST API. This makes the deployment of the model into production much easier. A user can train or make predictions just using the REST API.

3.3.1 Example

There is an example of usage explained in the article [14], where the Support Vector Machine (SVM) algorithm from scikit-learn [15] library was used. The model was trained on the IRIS [16] dataset. The IRIS dataset contains features measured on three types of Iris flower. The data consists of four columns – the length and the width of the sepals and petals, in centimetres. It is usually used for demonstrating purposes, where a machine learning model can predict a type of Iris flower based on those four features.

The prediction query is showed in the code snippet 3.2 a user sends four values – `data=5.1 & data=3.5 & data=1.4 & data=0.2"` and gets a prediction – `"labels": [ 0 ]`. The 0 indicates that the model predicts the flower to be of type 0.

Code snippet 3.2 Curl prediction query [14]

```
curl -s -X POST "http://127.0.0.1:5000/v2/models/iris-deepaas/
    predict/?data=5.1&data=3.5&data=1.4&data=0.2" -H  "accept:
    application/json" | python -mjson.tool
{
    "predictions": {
        "labels": [
            0
        ]
    },
    "status": "OK"
}
```

3.3.2 Summary

The DEEPaaS illustrates how the deployment of a model can be simplified using the Client-Server architecture. Any library from the python world can be used for the implementation of a machine learning model here. An application written in any language can handle the model using the REST API. The issue with model storing format is also solved here because almost every machine learning library supports export and import into and from a file.

3.4 Conclusion

This chapter showed how a flow of creation and deployment of an anomaly detection model goes in frameworks with a similar focus.

From EGADS could be taken a lesson about how anomaly detection is applied on the time-series data.

Another exciting part is in the Oracle solution, namely the PMML format for the model transfer. The fact that Oracle has been using this format confirms that it can be used in practice.

The last example of DEEPaas demonstrates how simpler it is to use a Client-Server architecture for model deployment.

Chapter 4
Technologies

This chapter will introduce the most used formats for storing a machine learning model and corresponding libraries for creating such a model, and libraries for scoring a model in Java runtime.

4.1 Model formats

Training of a machine learning model can be very time consuming, so there is a need to persist the result of training into a file. Each time someone wants to use a model, he only loads a model into memory from the file.

The information needed to be stored can vary per algorithm. For some models, it is enough to save coefficients calculated from during the training. But for some, it is needed to keep the training data. For the second case, the file size can increase rapidly with the size of the training dataset.

There are several formats in which we can store a model. Each format has its pros and cons. The most important characteristics are the memory needed for storage and how many tools support it. One of the not that important aspects is human readability, so you can quickly check what the actual model looks like without parsing the file and loading it into memory. This chapter will introduce the most commonly used formats.

4.1.1 Predictive Model Markup Language (PMML)

The PMML was introduced in 1999 by the National Center for Data Mining, the University of Illinois at Chicago, in the article [17].

This format is used by several R, Python or C++ machine learning libraries.

An essential library for this thesis is the JPMML library. JPMML provides parsing and evaluating of models stored in the PMML in Java programming language. More details about the JPMML are in the section 4.2.1 later in this document.

As apparent from [18], the PMML is an XML-based [19] machine learning model interchange format. More models can be saved into one PMML file and accessed by the name of the corresponding data mining function.

This standard supports algorithms such as a k-nearest neighbour, naive Bayes, linear regression... The whole current range of supported algorithms can be found on the Data Mining Group page (DMG) [18].

"*Certain types of PMML models such as neural networks or logistic regression can be used for different purposes. That is, some instances implement prediction of numeric values, while others*

Code snippet 4.1 PMML example – data fields [17]

```
<DataDictionary numberOfFields="6">
    <DataField name="petal␣length" optype="continuous" dataType="double"
        />
    <DataField name="petal␣width" optype="continuous" dataType="double"/
        >
    <DataField name="sepal␣length" optype="continuous" dataType="double"
        />
    <DataField name="sepal␣width" optype="continuous" dataType="double"/
        >
    <DataField name="species" optype="continuous" dataType="double"/>
    <DataField name="species_class" optype="categorical" dataType="
        string"/>
</DataDictionary>
```

Code snippet 4.2 PMML example – model header [17]

```
  <NearestNeighborModel modelName="KNN␣IrisGardens"
      continuousScoringMethod="average" categoricalScoringMethod="
      majorityVote" numberOfNeighbors="3" functionName="mixed">
```

can be used for classification. Therefore, PMML defines several different mining functions. Each model has an attribute functionName which specifies the mining function." [18]

4.1.1.1 Example

Here is an example of the k-nearest neighbour trained on the IRIS [16] dataset and stored in the PMML format. The whole example with detailed description can be found on web site [20].

XML part number 4.1 specifies input and target fields with their data types. Here it is four input fields and two target fields. One of the target fields is a categorical one meaning it predicts a category. The second one is continuous, which means it predicts a numerical value.

Essential information for creating the model is set in the 4.2 XML part. The `functionName` is "mixed", which means it produces categorical and predictive results at once.

Then a list of training instances follows 4.3. Instances are of the same shape as mentioned in the data fields section 4.1.

As visible in the XML snippet 4.4, the Euclidean distance was used in this example.

The algorithm takes four values as input. Calculates distance to each data point from the training instances and picks three nearest ones. The `species` field is then calculated as an average of the `species` fields from these three data points. The `categoricalScoringMethod` is "majorityVote". Therefore the field `species_type` will be the one which is the most occurrent in these three neighbour samples.

4.1.1.2 Anomaly detection model

Version 4.4 contains an anomaly detection model type. As mentioned in [21], the PMML anomaly detection model type can wrap a model of three kinds:

- Isolation Forest
- One Class support vector machine
- Clustering mean distance based anomaly detection model

Code snippet 4.3 PMML training instances [17]

```
<TrainingInstances recordCount="149" fieldCount="6" isTransformed="
    false">
  <!-- ... -->
  <InlineTable>
    <row>
      <sepal_length>4.9</sepal_length>
      <sepal_width>3.0</sepal_width>
      <petal_length>1.4</petal_length>
      <petal_width>0.2</petal_width>
      <target_species>10</target_species>
      <target_class>Iris-setosa</target_class>
    </row>
    <!-- ... -->
    <row>
      <sepal_length>6.3</sepal_length>
      <sepal_width>3.3</sepal_width>
      <petal_length>6.0</petal_length>
      <petal_width>2.5</petal_width>
      <target_species>30</target_species>
      <target_class>Iris-virginica</target_class>
    </row>
  </InlineTable>
</TrainingInstances>
```

Code snippet 4.4 PMML example – distance calculation method [17]

```
<ComparisonMeasure kind="distance">
  <squaredEuclidean/>
</ComparisonMeasure>
```

It is not many, but it is important to notice that this is more of a helping construct. A user can also handle anomaly detection with just a prediction mechanism saved in the PMML. A user can implement the anomaly indication mechanism comparing observed and predicted data outside of the PMML. Using the last approach extends the scope of usable algorithms for a user.

4.1.2 Portable Format for Analytics (PFA)

PFA, same as PMML, was developed by the Data Mining Group [22]. It is not massively widespread, unlike the beaten path of the PMML. It is more or less a scripting language. The language contains constructs such as local variables, user-defined functions (even lamdas), conditionals and loops.

"*A PFA document is a JSON document with additional constraints. The JSON content describes algorithms, data types, model parameters, and other aspects of the scoring engine. Some structures have no effect on the scoring procedure and are only intended for archival purposes. A PFA document is a JSON-based serialization of a scoring engine. A scoring engine is an executable that has a well-defined input, a well-defined output, and performs a purely mathematical task.*" [23]

In some ways, PFA is a successor to the PMML format. In my opinion, as the area of machine learning moves quickly forward, it is just a matter of till PFA replaces the PMML. The motivation for developing the PFA is described in the [24] article by its creators. They wanted to create an interchange format, which satisfies the following requirements:

- It is extensible by a user for adding new model types and pre- and post- processing logic.
- A user should be able to create a workflow by putting models into chains and hierarchy.
- "*The language should be easy to integrate into today's distributed and event-based data processing platforms, such as Hadoop [25], Spark and Storm[26].*" [24]
- It is safe to deploy – the deployed model does not have access to the IT operational environment resources or a network.

As further explained in the article [24], PMML lacks the first three requirements. When a user needs more algorithms or functionality, he has to ask a PMML working group to extend the standard by releasing a new version. In contrast, the PFA is a scripting language that gives a user a lot of freedom when creating the machine learning model. In the PMML, it is possible to store more models into one file, but a user still had to call just one evaluation function, which means he had to chain or hierarchize models under his direction outside of the PMML document.

4.1.2.1 Example

The very trivial example is shown in the code snippet 4.5. PFA uses the prefix notation for storing a calculation. The prefix notation means an operation symbol is before both operands in the expression. Let the input be a real number x, then an output y is calculated as $y = \sin(x+1)*2$. A more complex example with a description can be found here [24].

Code snippet 4.5 PFA simple example [24]

```
input: double
output: double
action:
- {m.round: {"*": [{m.sin:
{+: [input, 1]}}, 2]}}
```

4.1.3 Open Neural Network Exchange (ONNX)

Microsoft announced ONNX in September 2017 [27]. Microsoft and Facebook developed it together and published it as an open-source project. The name can be a bit misleading, as it is possible to store also different machine learning models than just a neural network. As mentioned on the ONNX website [28], this format represents a machine learning model as a graph of computation nodes. Further details about the internal technical design can be found on the GitHub page [29].

An ONNX machine learning model is saved as a serialized protocol buffer (protobuf) [29]. The advantage is that the serialization compresses the stored information. On the other hand, it is not human-readable. Therefore, the calculation logic is hidden from a person until it is loaded to a program run-time.

4.1.4 TensorFlow serialized protobuf

This custom format used by Tensorflow can store function written in TensorFlow. However, more worth noticing is the possibility to save a Keras [30] neural network. As described in [31], the algorithm is not stored in a simple document, but it is a whole directory with the following structure:

```
variables.....................................a folder containing a training checkpoint
  └── ...
assets..........................a folder containing files used by a TensorFlow graph
  └── ...
saved_model.pb.............file containing a model with defined signatures and functions
```

This format's pros and cons are the same as by ONNX – not human-readable vs compressed information. Another disadvantage is that the name of the function that a user wants to call is hidden. It is serialized, so a user must define his function name when saving a model and keep it somewhere. The second approach is when a user does not have access to the model creation and gets a completed model, he needs to deserialize the model and then look up his desired function in many functions.

4.2 Relevant libraries

Java is one of the most favourite programming languages nowadays. Many applications and projects are running on Java. It implies an intention to implement a way of how a machine learning model can be used directly in Java run-time. There already exists an implementation of a scoring engine for all formats mentioned in the section 4.1. It is a promising sign for the goal of the thesis because, there will not be a need to implement a scoring engine from scratch.

4.2.1 JPMML

Openscoring Ltd. company [32] with founder Villu Ruusmann created a whole set of libraries producing and consuming PMML models. The project is regularly updated and developed to provide more and more functionality to users.

4.2.1.1 Conversions to PMML

The creators of JPMML came up with two ways to implement the conversion into the PMML. The first way is to implement a package for a language on which the machine learning library

is running. The package can be embedded into the program and provides a saving of the model into PMML. The second option is to take a stored model in some format and implement the conversion in Java from the custom format into the PMML. On the overview below, there are listed machine learning libraries with corresponding ways of how the model created by the library can be exported into the PMML. The variety is comprehensive, and the list below does not contain all of them. To see the full JPMML capability, go to the JPMML GitHub website [33].

- R and Rattle:
  - JPMML-R library.
  - r2pmml package.
  - pmml and pmmlTransformations packages.
- Python and Scikit-Learn:
  - JPMML-SkLearn library.
  - sklearn2pmml package.
- Apache Spark:
  - JPMML-SparkML library.
  - pyspark2pmml and sparklyr2pmml packages.
  - mllib.pmml.PMMLExportable interface.
- H2O.ai
  - JPMML-H2O library.
- XGBoost:
  - JPMML-XGBoost library.
- LightGBM:
  - JPMML-LightGBM library.
- TensorFlow:
  - JPMML-TensorFlow library.
- ...

[34]

4.2.1.2 JPMML evaluator

JPMML evaluator is a scoring engine for evaluating PMML models in the Java run-time. Nowadays, it seems to be the most used library for evaluating PMML models in Java. JPMML implemented most of the functionality specified by the Data Mining Group standard. However, it is tough to keep up with the standard described on the Data Mining Group website, so it is not an exact representation, and some algorithms can be missing in the JPMML evaluator. Algorithms implemented in the JPMML evaluator are:

- Association rules
- Cluster model

- General regression
- Naive Bayes
- k-Nearest neighbors
- Neural network
- Regression
- Rule set
- Scorecard
- Support Vector Machine
- Tree model
- Ensemble model

[34]

4.2.2 PFA Hadrian

Hadrian is a scoring engine for evaluating a PFA model. As described on the GitHub page, Hadrian is intended to be utilized as a library embedded into the application running on the Java Virtual Machine (JVM) or used as a scoring engine container.

The Hadrian repository also contains two other independent libraries – Titus and Aurelius.

4.2.2.1 Titus

Titus library is a complete implementation of the PFA for Python. It is more focused on developing the model rather than scoring it in Python. As mentioned on the Titus wiki page [35], it can run only on Python versions between 2.6 and 3.0. An alternative to the Titus can be the Titus2 [36], a fork of the original Hadrian repository. Titus2 declares to be compatible with newer versions of Python.

Titus uses a PrettyPFA parser for easier creation of the PFA document. The PrettyPFA parses a string containing the code written in the PrettyPFA custom scripting language and converts it to the PFA format. When using the PrettyPFA, a user does not need to write everything in a prefix notation, as previously mentioned in the PFA section 4.1.2.1. He can formulate the code in a more programmer-friendly language. In the code snippet 4.6, it is shown how the quadratic equation can be written in the PrettyPFA.

4.2.2.2 Aurelius

Aurelius is a subproject with a similar goal as Titus but in the R programming language field.

4.2.3 ONNX-runtime

ONNX-run-time is a scoring engine for ONNX models. It is available for Windows, Linux and Mac. ONNX-runtime implemented the scoring machine for nearly all popular programming languages. As introduced in the official Youtube video by Microsoft [37], worth noticing is that it runs with CPU or GPU with an extensible architecture that can plug in additional hardware that accelerates computations.

The most important fact for the thesis is that the ONNX-runtime supports the model evaluation in Java run-time. A lovely bonus is that the ONNX-runtime supports the computing

Code snippet 4.6 PrettyPFA example [35]

```
>>> pfa = prettypfa.json('''
input: record(a: double, b: double, c: double)
output: union(null,
              record(Output,
                     solution1: double,
                     solution2: double))
action:
  var a = input.a, b = input.b, c = input.c;

  var discriminant = b**2 - 4*a*c;
  if (discriminant >= 0.0) {
    // if there are any real solutions, return them
    var x1 = -b + m.sqrt(discriminant)/(2*a);
    var x2 = -b - m.sqrt(discriminant)/(2*a);
    new(Output, solution1: x1, solution2: x2)
  }
  else
    // otherwise, return null (N/A)
    null
''')
```

power management in the environment where it is running. It can be configured to use CPU, GPU or FPGA for executing calculations. In code snippet 4.7, the GPU was used for the model evaluation.

Code snippet 4.7 ONNX GPU example [38]

```
    int gpuDeviceId = 0; // the GPU device ID to execute on
    var sessionOptions = new OrtSession.SessionOptions();
    sessionOptions.addCUDA(gpuDeviceId);
    var session = environment.createSession("model.onnx", sessionOptions
        );
```

4.2.4 TensorFlow

"*TensorFlow Java can run on any JVM for building, training and deploying machine learning models.*" [39] The TensorFlow core runs on the C++ backend, but it supports several languages other than only C++. Tensorflow is mainly known for its python package for training machine learning models. However, TensorFlow decided to support also programming languages such as Java, JavaScript, or Go. The TensorFlow Java library provides evaluation of a function stored in the serialized protobuf. The most important information for the thesis is that it implements parsing and evaluating Keras neural network models stored in the serialized protobuf file.

The ONNX and Tensorflow have some parts in common, namely saving the model into the serialized protobuf format as a graph and supporting execution on GPU.

Chapter 5
Design

This chapter presents the proposed design of the anomaly detection framework. The planned way of utilizing the framework is caught in the 5.1 section. The class diagram and discussion about the solution can be found in the 5.2 section.

5.1 Activity diagram

This section describes the planned flow of an anomaly detection model's usage. The flow is showed in the activity diagram 5.1.

5.1.1 Loading

There are two options of how to provide a model into the framework. The first one is to implement the model in the Java programming language. The second way is to provide a trained model as a file. The model is parsed from the file and loaded into the Java run-time represented as an instance of a class implementing the `IAnomalyDetectionModel` interface. The loaded model then can be put into a collection that gathers models into one place. A user of the framework decides whether he uses the loaded model directly or will, for example, firstly parse and load all of the models, put them into the collection and then takes the models he needs.

5.1.2 Feeding

After a successful load, it comes to real-time feeding the model with data that the user wants to analyze. The user analyzes a small batch of the data and sets a timeout for the calculation. The batch is a collection of data points containing a vector of features. The fed model can update its internal state based on the input. The update can mean retraining in real-time, but depending on an algorithm used in the model, it might be time-costly or even impossible to retrain.

The importance of timeout is evident. The calculation time is not the same for every input, and it can vary depending on the input's size and shape. The user avoids the neverending calculation by setting a timeout making the feeding more robust and fluent.

The output of each iteration is a set of detected anomalies. The implementation of the model can hold some internal memory about the previous input. Therefore, returned outliers are not necessarily pointing at a subset of the batch, but it can be any part of the data that had gone through the model.

5.1.3 Saving

As already mentioned, the model can keep and update its internal state. For this reason, it makes sense to provide a possibility to save the model affected by the real-time flow. However, in my opinion, it is just a side feature because, for most of the models, it will not be applicable. The first argument is that it would be needed to implement the conversion into the file. Also, the framework is not focused on training. Still, if some model suitable for storing occurs, the possibility of implementing a saver is there.

5.2 Class diagram

This section introduces the concrete class representation of the implemented framework. The diagram 5.2 contains necessary parts of the framework. Though the class diagram is not covering all of the code written, it clarifies the main idea and introduces the most important classes. More inside about the source code is introduced in the implementation section.

5.2.1 IModelParser

The `IModelParser` interface parses essential data from a file, string or stream and creates the `IAnomalyDetectionModel`. It is needed to implement at least one of these methods to realize the interface a sane way.

5.2.2 IModelSaver

The only `IModelSaver`'s responsibility is to save the model into a file. Even though the `IModelParser` may implement the conversion from string and stream, the corresponding operation by `IModelSaver` is, in my opinion, redundant as it would not be most probably used.

5.2.3 IModelConvertor

Both `IModelSaver` and `IModelParser` extend the `IModelConvertor` interface, requiring its descendants to implement the methods returning the model's format and name of the algorithm. `IModelConvertors` then can be looked up using the combination of these two fields.

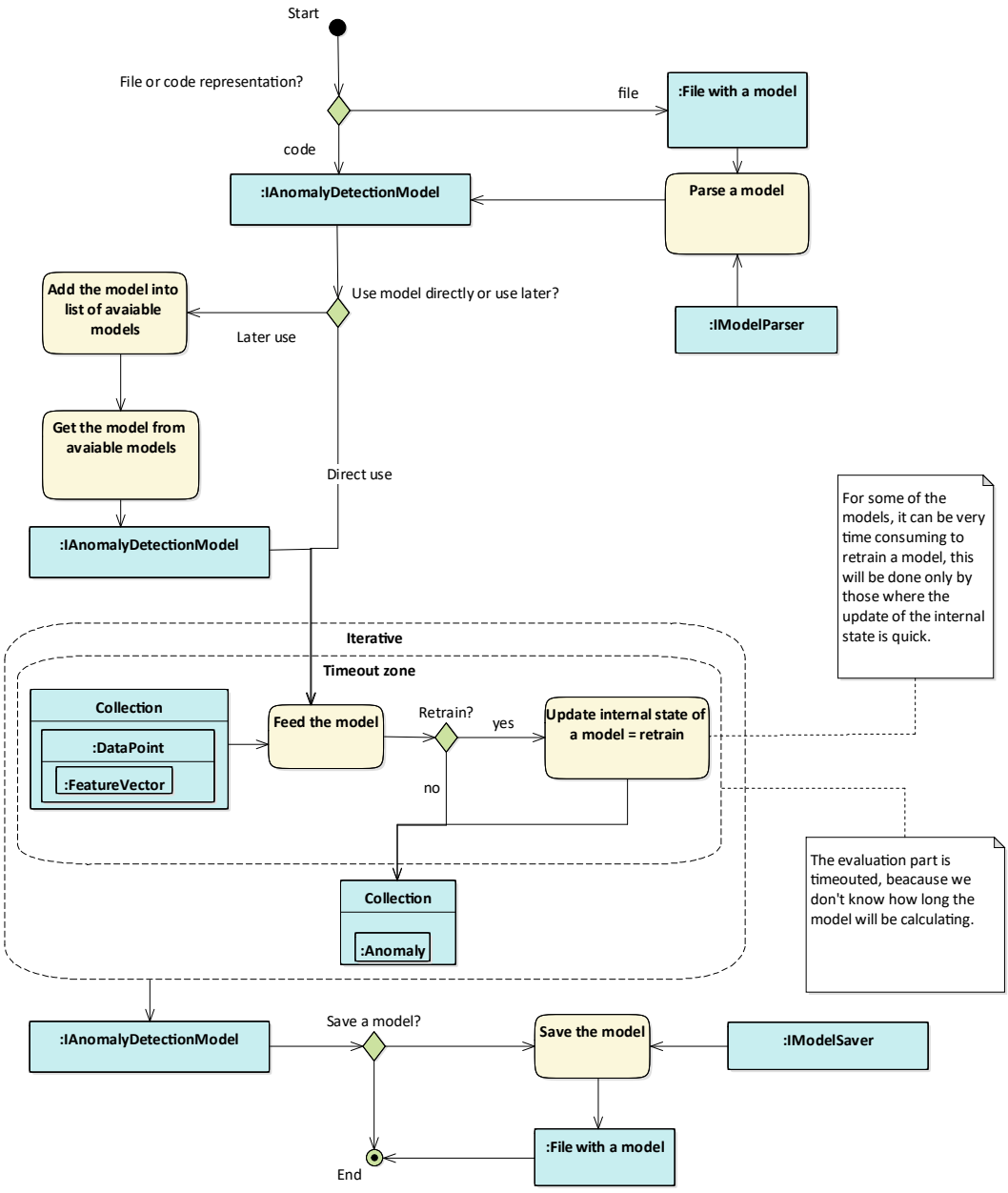
5.2.4 IAnomalyDetectionModel

The `IAnomalyDetectionModel` has a pretty simple interface. The model takes a list of data points and returns a list of anomalies. For now, the outliers are just of the point type (not contextual or collective). The framework can be extended in the future to distinguish between the kinds of anomalies. Please see the chapter 1.1 for more inside about the model's output.

5.2.5 IModelInputMetaData

The `IAnomalyDetectionModel` holds metadata about the shape of input, namely the range specifying an allowed size of the input list and information about the types of each feature in one data point. The primary purpose of keeping the metadata is for input validation.

Figure 5.1 Model life cycle



5.2.6 IIdentifiable

The `IAnomalyDetectionModel` extends the `IIdentifiable` interface to be searched for in some collection by the key. An observant reader may ask why the model is not identified by format and name as by `IModelParser` or `IModelSaver`. The reason is that a user may load a model

from some format but then saves it in a different format than initially taken.

5.2.7 DataPoint

The `DataPoint` holds a set of features and id. `Anomaly` wraps a `DataPoint` and adds a report such as the probability that the `DataPoint` is an outlier. Also, the `DataPoint` can be extended. For example, when analyzing a time series, the crucial characteristic is the `TimeStamp` when the data instance was observed. It makes sense to separate the `TimeStamp` as a separate field in the `TimeSeriesPoint` class and use it as a unique identifier.

5.2.8 Feature

The feature is represented as a `Feature` class. The most important is to get the type and value of the feature. The name helps a user to recognize the feature.

5.2.9 IFeedingSession

As outlined before, the feeding of the model needs to be handled. The interface responsible for it is the `IFeedingSession`. `IFeedingSession` wraps the `IAnomalyDetectionModel` instance. The main motivation is to apply the timeout to the calculation. The feeding method is overloaded so that the task can run with or without the timeout.

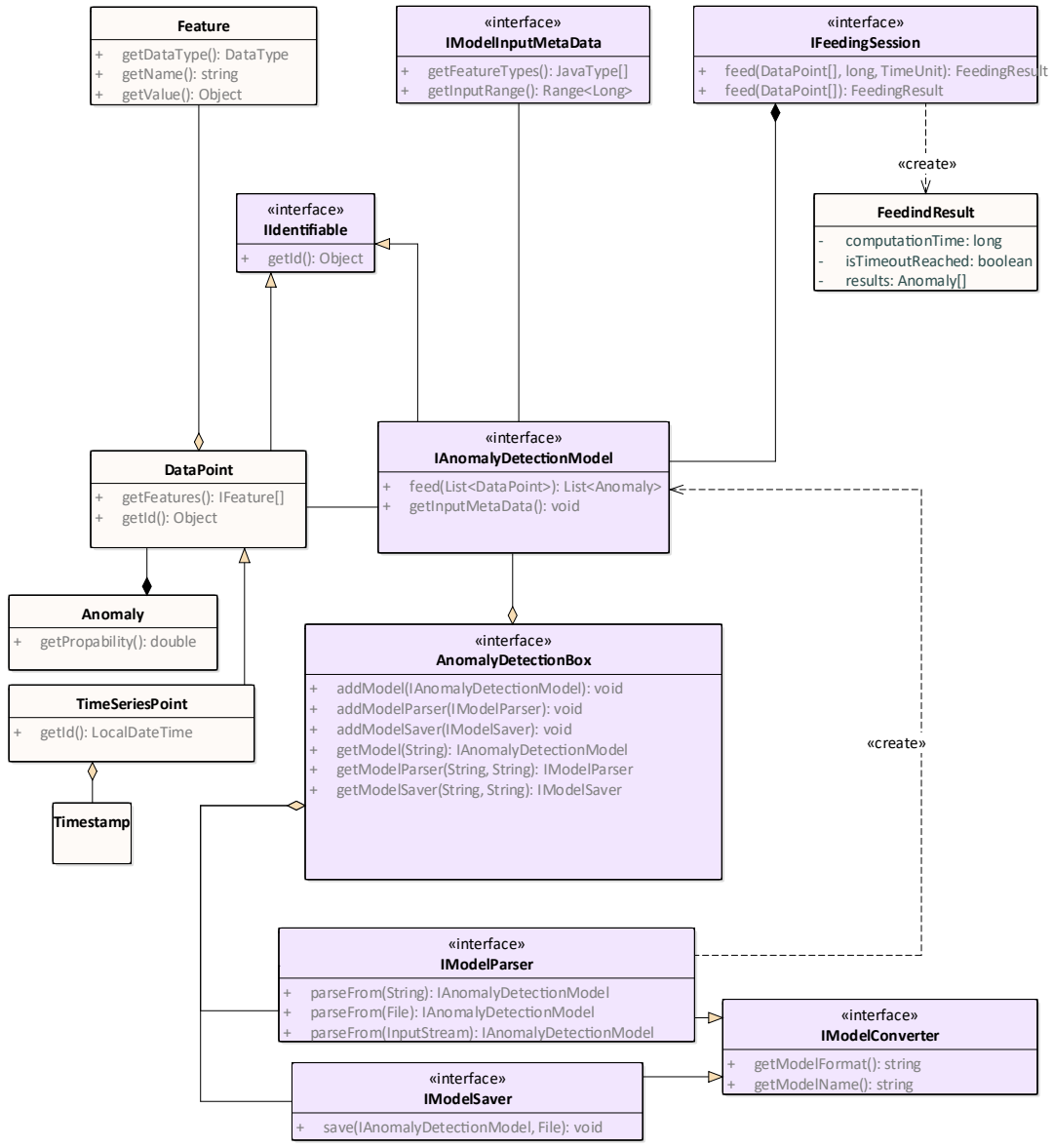
5.2.10 FeedingResult

The product of a feeding process is the `FeedingResult` instance containing a list of anomalies and additional info about the computation – computation time in nanoseconds and a flag about reaching the specified timeout.

5.2.11 IAnomalyDetectionBox

The `IAnomalyDetectionBox` gathers together implemented `IAnomalyDetectionModels`, `IModelSavers` and `IModelParsers` at one place. Models are looked up using the id, and parsers and savers utilizing the combination of model format and algorithm name.

Figure 5.2 Class diagram



Chapter 6
Realization

This chapter shows the implementation in more details. The skeleton from the design part has been implemented. Besides, it has been enriched with supporting classes.

I have created an abstract class for most of the interfaces. A user of the framework can decide whether the corresponding abstract class suits his needs or he wants to implement methods from scratch.

Firstly the classes representing models and their hierarchy are described in the 6.1.2 section. Then comes info about the classes representing data in the 6.1.3 section and handling the model's evaluation in the 6.1.4 section.

6.1 Implementation

6.1.1 IntelliJ UML diagram legend

Even though the diagram is quite intuitive, it makes sense to declare the legend for symbols used in diagrams created in IntelliJ IDEA [40]. The whole legend can be found in the IntelliJ documentation [41]. In the following list, there are clarified the most frequently used symbols in an IntelliJ UML diagram:

(m) – method

(f) – field

(p) – property

(i) – interface

(c) – class

Unlocked padlock – public

Locked padlock – private

Key – protected

Otherwise, the IntelliJ IDEA follows the UML [42] conventions.

6.1.2 Classes representing model

As stated in the theoretical section 1.1.2 about the anomaly detection workflow, the anomaly detection model may consist of some predictive mechanism which's output is then processed by some logic comparing the predicted and observed data and labelling outliers based on the error.

For this reason, I have decided to create the `IPredictiveModel` interface – model, which will produce a list of predictions rather than outliers itself. the `IPredictiveModel` could be then used as a component of the `IAnomalyDetectionModel`. Both `IPredictiveModel` and `IAnomalyDetectionModel` have something in common.

Both can be fed with a list of data points and returns a list holding `Prediction` – for the prediction model – or `Anomaly` – for the anomaly detection model. This led to creating the `IModel<T>` interface, where `T` is a generic parameter specifying the type of returned objects in the list. It makes sense for each model to keep the metadata about the input shape. The abstract implementation of the `IModel<T>` is `AbstractModel<T>`.

The `AbstractModel<T>` contains the feeding logic described in the figure 5.1. the whole process is shown in the code snippet 6.1. Furthermore, it applies an `IInputValidator` onto the incoming data. If the input is considered wrong, then the `IInvalidInputHandler` is called to handle such a situation. The `IInputValidator` and `IInvalidInputHandler` instance is passed as an argument in the constructor. The motivation is the following: a user can configure the intensity of the handling of the incorrect input. After the input validation comes update of an internal state and producing an output.The `AbstractModel#updateInternalState(List<DataPoint>)` and `AbstractModel#produceOutput(List<DataPoint>)` methods are abstract.

Code snippet 6.1 AbstractModel#Feed method

```
public final List<T> feed(final List<DataPoint> dataPoints) {
    try {
        inputValidator.validate(dataPoints, inputMetadata);
    } catch (final InvalidInputException e) {
        invalidInputHandler.handle(e);
    }
    updateInternalState(dataPoints);
    return produceOutput(dataPoints);
}
```

6.1.3 Classes representing data

The following text is describing the logic depicted in the 6.2 diagram.

The classes making the most significant part are Feature and `DataPoint`. `DataPoint` contains a list of `Features` and an Id. For the `TimeSeriesPoint`, the `LocalDateTime` [43] represents the unique identifier. For the plain `DataPoint`, the Id can be, for example, some index.

The `Feature` contains name, `IFeatureType` and some value.

The `IFeatureType` interface provides a name of the type and `Set` of possible Java classes for the value. The first implementation of the `IFeatureType` is an enum providing all primitive types. The second implementation is a simple class. If a user does not find the desired type in the enum range, he can create his custom one using the `FeatureType` class.

For testing and demonstrating purposes, there have been implemented printers for Feature and `DataPoint` classes. The visitor pattern is applied here to achieve the single responsibility principle. Both printers are utilizing the Gson library [44] for converting objects into JSON format.

`Anomaly` class wraps a `DataPoint` instance and adds additional info such as the probability that the point is an outlier.

Figure 6.1 Model hierarchy

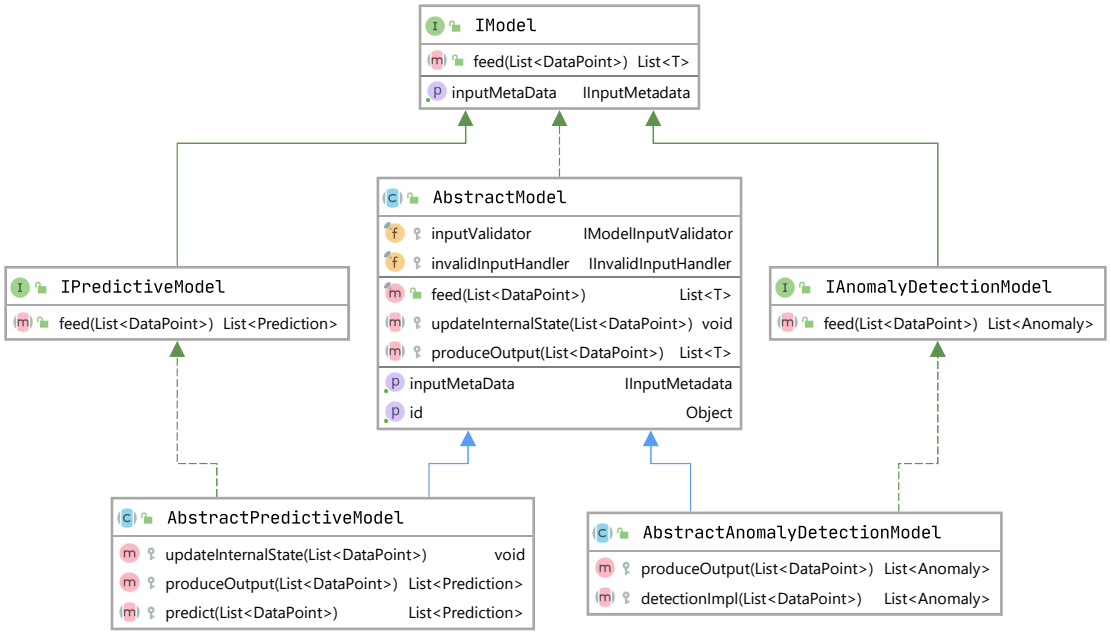
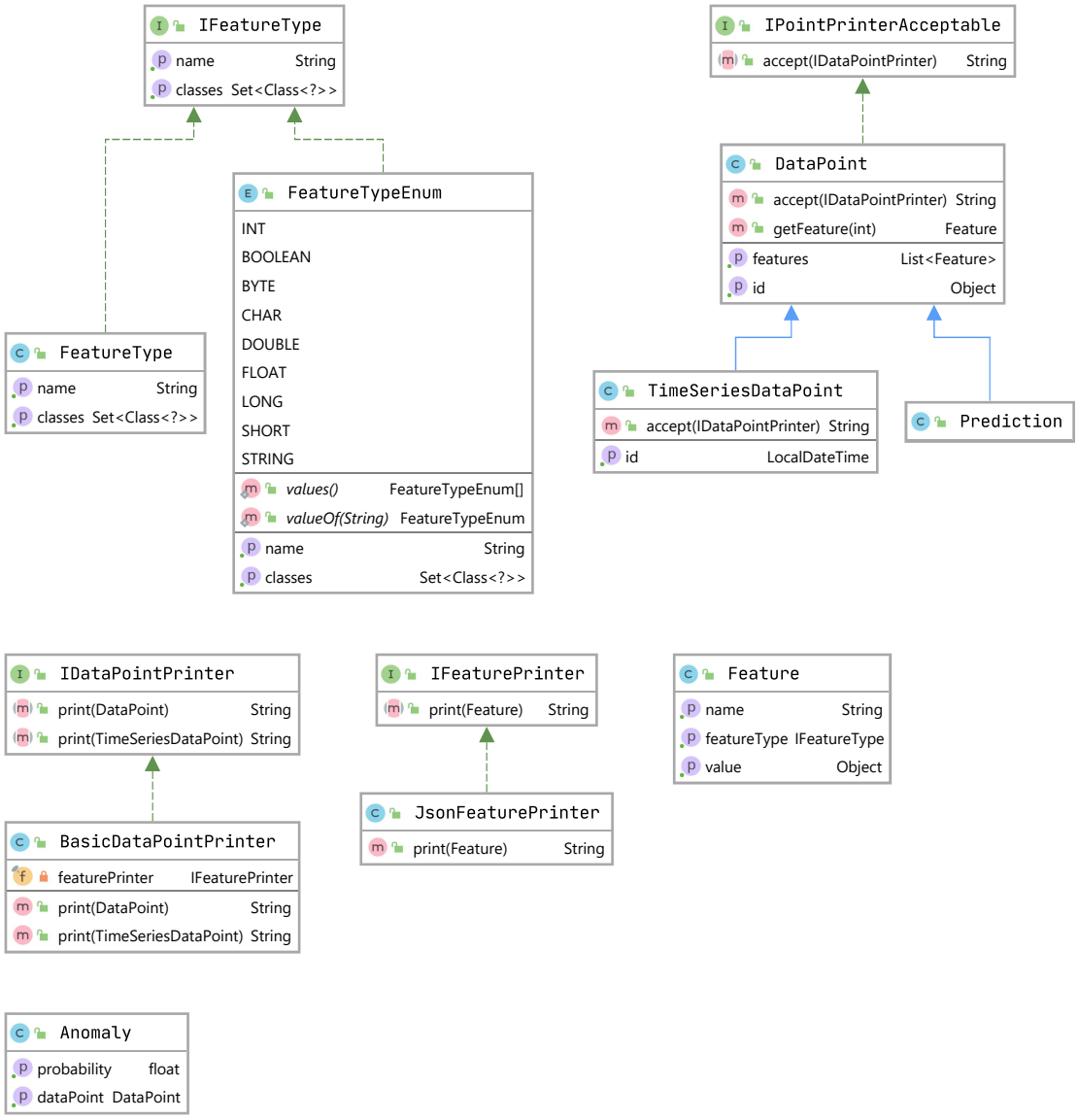


Figure 6.2 Classes representing data



6.1.4 Classes handling an evaluation of a model

The following text is describing the logic depicted in the 6.3 diagram.

The abstract implementation of the `IFeedingSession` – `AbstractFeedingSession` consists of the wrapped `IModel` and `TimeoutService` instance managing the task's run with a timeout. The `TimeoutService` takes a `Callable` [45] and runs it with a timeout. There was created an implementation for the `TimeoutService`. The implementation – `FutureTimeOutService` class – utilizes the `Future` [46]. The product from running a task in `TimeoutService` is an instance of the `TimeoutResult` – inner class of the interface – containing the output of the calculation, time spent on the calculation and a flag about reaching the timeout.

The final outcome of the session's feeding is `FeedingResult` class containing a list of objects of type `T`, where `T` is the type of objects produced by the `IModel` instance and again info about the calculation time a flag about reaching the timeout.

6.2 Testing

The code is richly covered with unit tests, ensuring all classes are working as they should. Junit [47] framework was used for writing unit tests.

Another tested area is that loaded models return the same output as the model running in the environment where it was created initially. It is again in the form of Junit tests, but now it is testing the whole flow – from loading the model, feeding the model and comparing the returned results. The testing of examples of models is delineated in the chapter with examples of usage.

6.3 Documentation

The thesis document itself contains a description of implementation and design, giving main ideas and concepts to a reader. Worth documenting is the source code itself and a tutorial of how a user can embed the framework into the existing Java project.

As the resulting software does not have any graphical user interface, there is no need to create an end-user manual.

The source code is filled with Javadoc [48] comments from which has been generated the Javadoc documentation explaining the code in every detail.

6.3.1 Embedding of the framework

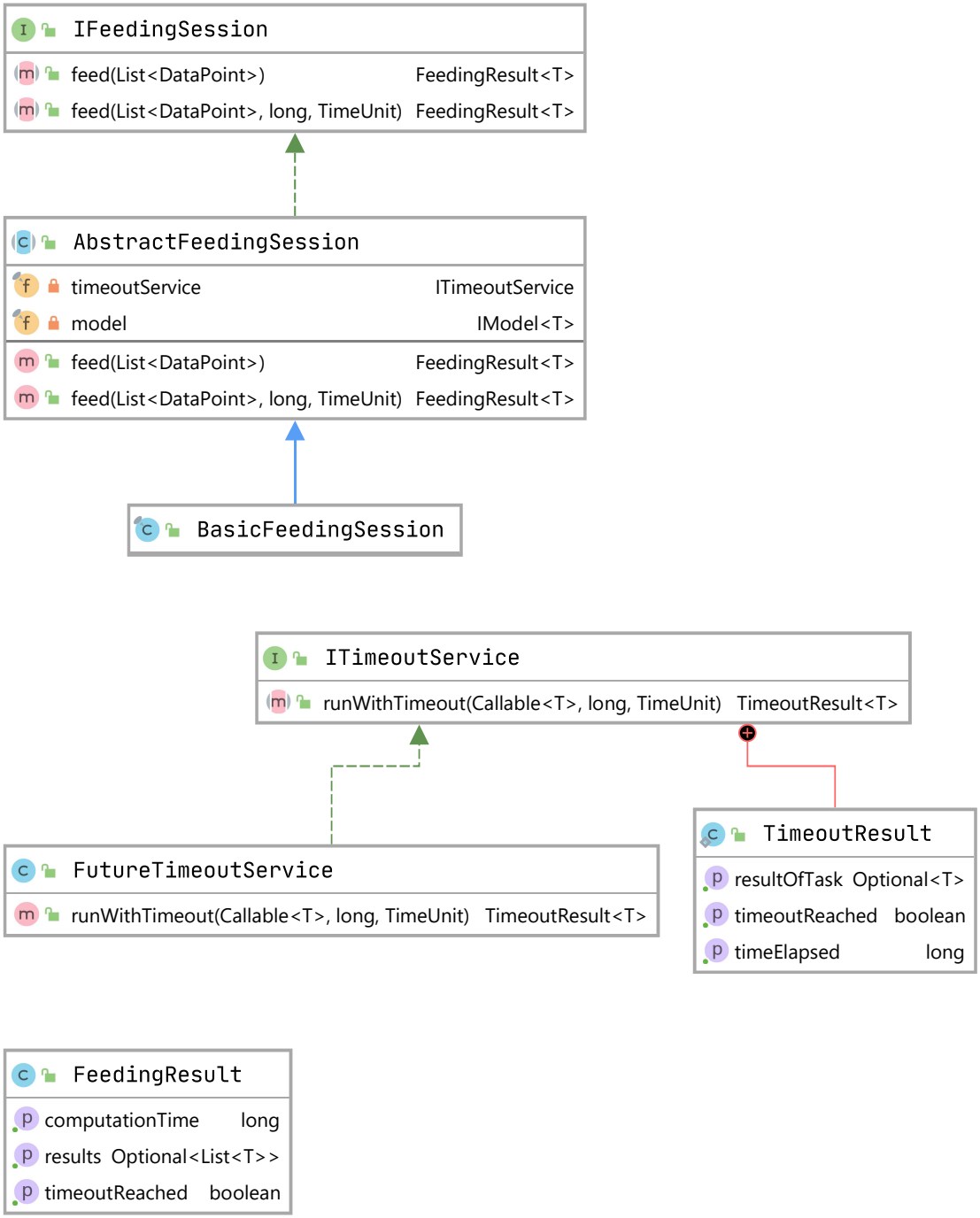
The framework is intended to be used as a package or a module in an existing Java project. This section demonstrates the integrating of the framework.

6.3.1.1 IntelliJ IDEA – import as a module

This section will show how you can import the project into a project in IntelliJ v. Community addition 2020.2. The simplest way is to clone the repository and import a module into your existing Java project.

1. Clone the repository to your local machine.
2. Open the project you want to import into.
3. Go File –> Project Structure –> Modules.
4. Click on the add button.
5. Select import module.

Figure 6.3 Feeding session hierarchy



6. Select the whole cloned repository with the Anomaly Detection Framework and press ok.
7. After the last step, you can add a dependency to the module you want to use classes from the framework.

6.3.1.2 Copy&paste as a package

If the first option does not work for you, you can do it a bit more dummy, but still working way. Again clone the repository to your local machine and copy&paste the code to your project. You will need to resolve library dependencies and other stuff.

Chapter 7
Examples

7.1 Used theory

7.1.1 K-Nearest neighbour algorithm (KNN)

This subsection sources from the presentation [49]. The problem can be imagined as estimating a value of $y \in R$ for a given $x \in R^p$, where training data is $X \in R^{N,p}$ and known dependent variables $Y \in R^N$. The main idea of the algorithm is to find k nearest points using some distance metric. The distance metric can vary, the most used and most intuitive one is the Euclidian distance. After finding the neighbours there are two ways of predictions – classification and regression. The classification predicts $y \in Y$ as the most frequently occurred y value of its neighbours. The regression predicts y as an average of y values of its neighbours.

7.1.2 Recurrent neural network (RNN)

This subsection sources from [50]. The RNN is a special type of artificial neural network processing sequential or time-series data. The difference to the feed-forward neural network, which considers inputs and outputs independent, is that RNN takes into account the output of the previous calculation and current input. The flow is depicted in the diagram 7.1. Assuming X_t is input and Y_t is output in time t, the network can be unrolled and presented as shown on the right part of the diagram 7.1.

The input and output side is not always one to one. It is possible to, as shown in the diagram 7.2, provide many inputs and taking just one output.

7.1.2.1 Long short-term memory (LSTM)

LSTM is one of the RNN architecture. The details about the LSTM are not necessary to comprehend. The main point is that the neural network adjusts the prediction based on the memory about the previous input.

7.1.3 Autoencoder architecture

This subsection sources from [51]. The autoencoder architecture consists of two processes – encoding and decoding. The input vector $x \in R^N$ is encoded to the vector $y \in R^M$ where $M < N$. From input were extracted the most important characteristics. Then comes the

Figure 7.1 RNN rolled [50]

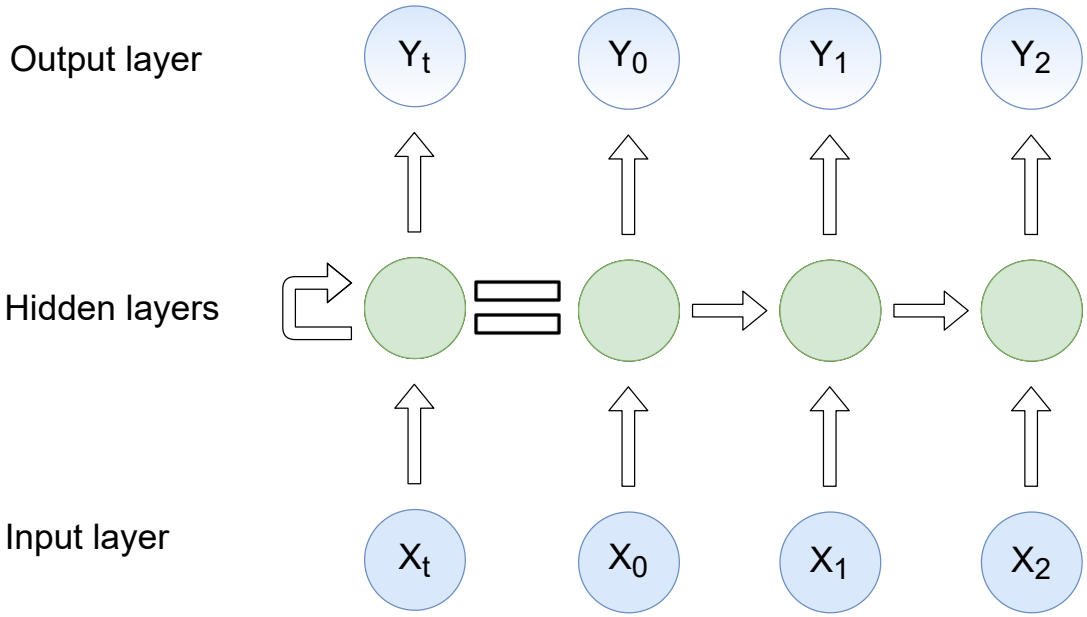
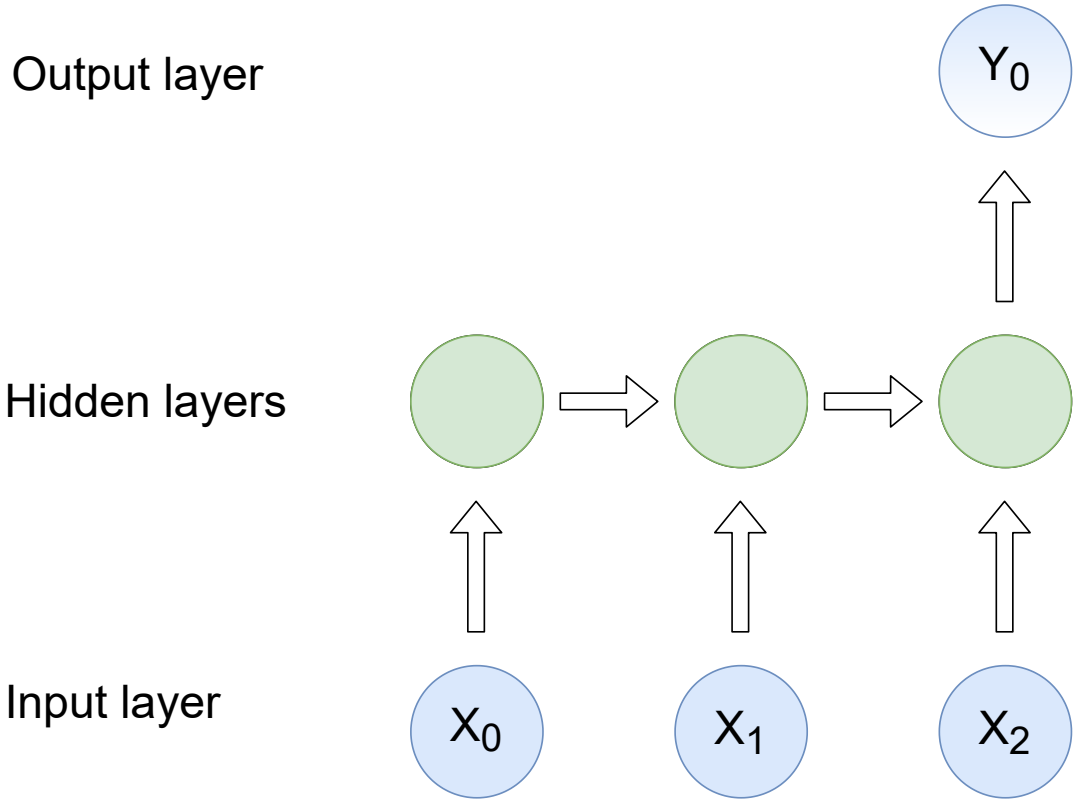


Figure 7.2 RNN many to one [50]



decoding of the vector y to vector $z \in R^N$. This technique is commonly used for removing noise from data or making predictions.

7.2 PMML

This example is using the model stored in the PMML format. This whole section sources from the Python notebook published on the Kaggle website [52]. The python notebook in the attachment contains many algorithms used on the same dataset. I chose the KNN algorithm predictive model because it is one of the models supported to convert into PMML using sklearn2pmml library and parsed using JPMML library.

The model predicts a future average price of one piece of avocado using the KNN. A data point is an anomaly once the actual price differs too much from the predicted one. I used here a simple threshold to identify outliers. When the error between the real and the forecasted value is above the threshold, it is an anomaly.

The training data set contains weekly retail scan data coming directly from retailers' cash registers between 2015 and 2018.

The model needs the features mentioned below to be able to predict the average price.

Type – conventional or organic way of growing

Region – the city or region of the observation

Total Volume – total number of avocados sold

Small Hass – total number of avocados with PLU 4046 sold as one piece

Large Hass – total number of avocados with PLU 4225 sold as one piece

XLarge Hass – total number of avocados with PLU 4770 sold as one piece

Bag with Small Hass – total number of avocados with PLU 4046 sold as four pieces in a bag

Bag with Large Hass – total number of avocados with PLU 4225 sold as four pieces in a bag

Bag with XLarge Hass – total number of avocados with PLU 4770 sold as four pieces in a bag

Hass is a type of avocado. By mentioning Hass, it is meant one piece of the fruit. In a bag, there are four pieces

The dataset needed preprocessing before training the model with it. The type and region are slightly different from other columns because they had been stored as strings, not as a number value. The author of the python notebook created a new column for each value from type and region columns. Value 1 indicates the presence of the region or a type used for this data record.

The the simple fction depicted in the code snippet 7.1 does the transformation mentioned above.

Code snippet 7.1 Saving KNN model using sklearn2pmml [52]

```
pd.get_dummies(X[["type","region"]], drop_first = True)
```

The original dataset contains a lot of records. The author of the python notebook split the dataset into the training and testing part with 10,172 samples for training and the rest starting from 10,172 for testing. I took a different ratio because the trained model consumes circa (number of the columns) * (number of rows) lines in the file, which means that the model with 50 features and 10,172 records is stored into a PMML file with more than 500,000 lines. The parsing of such

a model takes unnecessarily a lot of time. For testing purposes, it is acceptable to have a smaller training dataset with pros, that the actual load of the model is significantly faster. I took 1,000 samples for the training part, making the resulting PMML file have 50,000 lines.

I saved the model using the sklearn2pmml library similar way it demonstrated in the tutorial [53] of sklearn2pmml.

Code snippet 7.2 Saving KNN model using sklearn2pmml

```
from sklearn2pmml.pipeline import PMMLPipeline

pipeline = PMMLPipeline([
("regressor", Knn)
])
pipeline.fit(X_train, y_train)

from sklearn2pmml import sklearn2pmml

sklearn2pmml(pipeline, "KnnAvocadoReg.pmml", with_repr = True)
```

Then I parsed the stored model using the JPMML library to use the model in the Java code. I created a unit tests checking first 50 predictions are the same as when running the model in the python notebook.

7.3 TensorFlow

The TensorFlow library supports creating a Keras [30] neural network and storing it into a file. In both examples below, an RNN with LSTM and autoencoder architecture is used. As mentioned in the theory section 7.1.2, the outstanding characteristic of such a model is that it considers the previous input when predicting, making predictions more accurate.

7.3.1 S&P 500 index

This example was taken from the website [54] and corresponding YouTube tutorial [55]. In this example, the author of the model tried to detect anomalies on the time series dataset with only one feature – S&P 500 index from 1986 to 2018. "*The S&P 500 Index, or the Standard & Poor's 500 Index, is a market-capitalization-weighted index of the 500 largest publicly-traded companies in the U.S.*" [56]

The dataset contains 7,752 rows with just two columns – date and closing price for the day. I took the same split for training data as the author of the tutorial, 95 % for the model's training, and the rest 5 % for testing. The dataset is ordered by date. The finalised model accepts 30 data points as an input and predicts the same count of data points as output. Then the mean absolute error is calculated between the input and output batch. The input batch is considered abnormal if the mean absolute error is above the specified threshold. For simplicity, only one data point is marked as an outlier – the point representing the first following day after the input batch's last day. For example, when 31st January is tagged as an anomaly, then the epoch from 1st January to 30th January is abnormal.

I saved the model using the Tensorflow API for the Python language. Then I took the folder with the saved model, loaded it using the Tensorflow API for the Java language. I fed the model with the testing data and compared Java results to the one from the python notebook (.ipynb). I created a unit test checking that the same data points were marked as an anomaly.

7.3.2 Message rates

The example, taken from Martina Fuskova's thesis [57], is about detecting anomalies in the electronic trading data, namely in the message rate data set. The anomaly detector, implemented by Martina, was used for getting generating the preprocessed data. Also, the model trained on the data was stored into a file and taken.

There is a number of messages sent per time window specified for each message type. The rates are store in the db as shown in the snippet 7.3.

Code snippet 7.3 Extracted message rates [57]

```
...

begin,end,RejectsAll, ..,MessageRateNewOrderSingle, ...
20180202-07:40:00.000,20180202-07:44:59.999,0, ...,0, ...
20180202-07:45:00.000,20180202-07:49:59.999,0, ...,32, ...
20180202-07:50:00.000,20180202-07:54:59.999,0, ...,8, ...
20180202-07:55:00.000,20180202-07:59:59.999,0, ...,16, ...
20180202-08:00:00.000,20180202-08:04:59.999,0, ...,544, ...
20180202-08:05:00.000,20180202-08:09:59.999,0, ...,368, ...

...
```

Those rates were extracted from FIX logs showed in the code snippet 7.4. I chose the NewOrderSingle message rate for anomaly detection. The anomaly detector user can specify the column for anomaly detection inside the config file. The time window is five minutes long. Values are scaled in the range 0 to 1.

Code snippet 7.4 FIX log [57]

```
20180529-09:31:33.145 : 8=FIX.4.2|9=266|35=D|49=00000702|...
20180529-09:31:34.163 : 8=FIX.4.2|9=355|35=8|34=21010|...
20180529-09:31:36.163 : 8=FIX.4.2|9=393|35=8|34=21011|...

...
```

The used model is of the same type (LSTM autoencoder) as in the first example. The model accepts just one data point and predicts one back. There is only one feature in the data point – scaled message rate (in range 0 to 1) in a five-minute time window.

Martina used more complex mathematical methods for detecting anomalies utilizing the knowledge of the whole actual and predicted time series. For the sake of simplicity, I used a simple threshold for identifying outliers, the same way as in previous examples. This approach's advantage is that it is possible to determine that a particular data point is an anomaly using just this data point (not a lengthier part of the time series).

Here are my results using the simple threshold shown in the graph 7.3. The result when using more advanced methods than just a simple threshold is represented in the graph 7.4. Anomalous points are manifested as a red circle. The predicted scaled value is calculated to represent the message count as follows: $y = x * max$, where x is a predicted scaled value, max is a maximal message count from the whole series and y is the resulting predicted message count.

I created a unit test checking that the predictive model returns precisely the same scaled results as the one from the Martina's anomaly detector written in Python.

Figure 7.3 Anomalies detected using the simple threshold

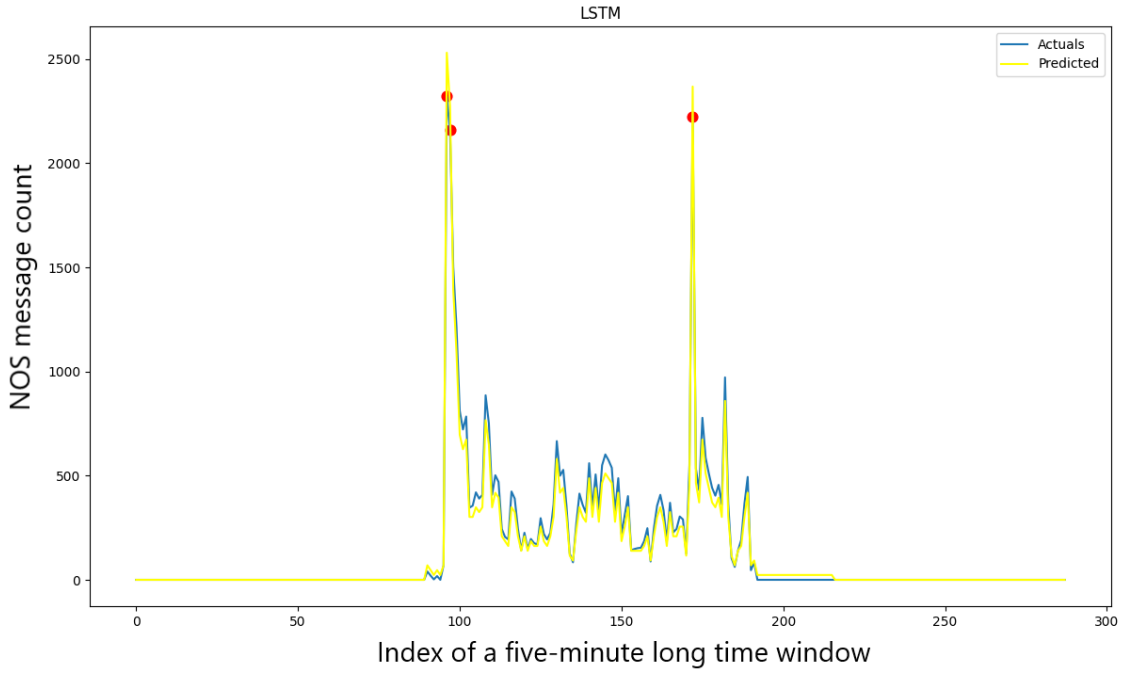
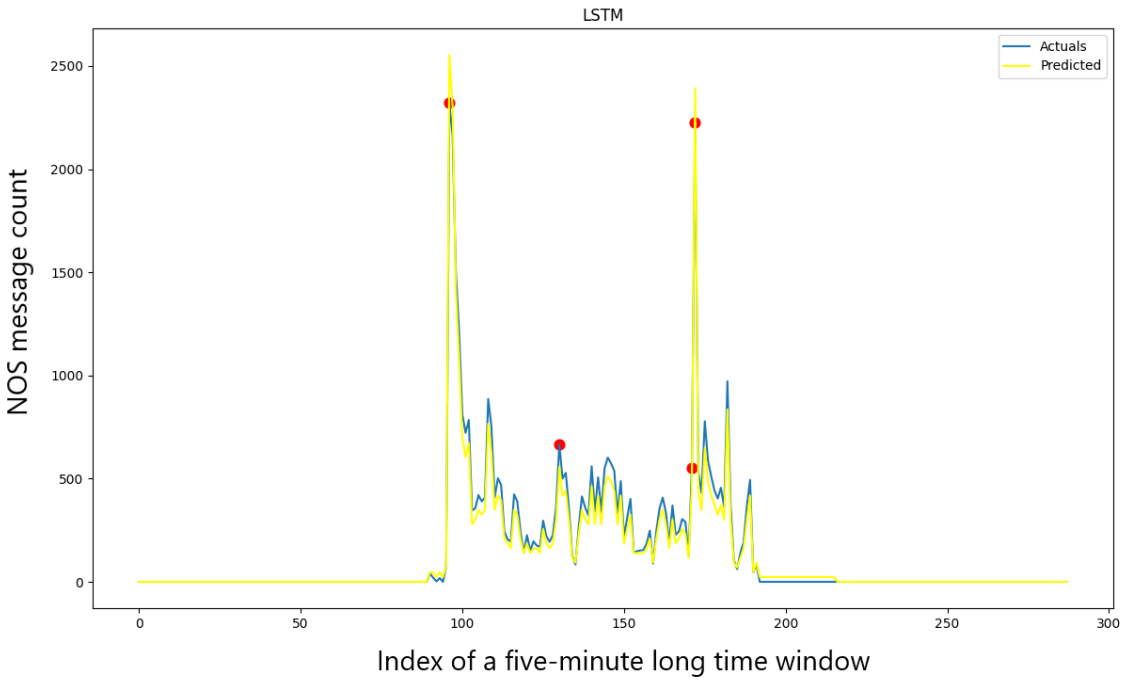


Figure 7.4 Anomalies detected using Martina Fuskova's comparison



Chapter 8
Conclusion

The main goal was to implement an anomaly detection framework, which will load anomaly detection models stored in a file in some format and use them in Java run-time on the incoming data in real-time. The goal has been fulfilled.

The main product of the thesis is the implemented extensible framework scoring the model in Java run-time in real-time. The framework is designed to add new formats and algorithms easily. It has been written purely in the Java programming language. The functionality is demonstrated on examples of usage of a PMML and TensorFlow's custom format. The source code is covered with unit tests and documented in Javadoc, describing each class in detail and UML diagrams, explaining the main idea of each critical part of the framework. The way the framework is intended to be used is also explained.

The essential step to successfully create the desired functionality was to get familiar with the machine learning world. That means to research frameworks with a similar focus and commonly used formats for storing models, machine learning libraries for the export/import of this model into/from the corresponding file format.

Also, there has been done relatively detailed research about nowadays used formats for storing a model and corresponding libraries for creating models and running them in a Java run-time. The very positive finding is that there already exist ways of loading and evaluating machine learning models in Java run-time.

8.1 Future work

The upcoming action is to embed the framework into the customer's existing software. Moreover, there are several ways worth working on in the future. Ideas and suggestions are presented in this section.

8.1.1 Extend a supported range of formats and algorithms

The most evident one is implementing support for chosen formats and algorithms. The step preceding the implementation could be research about benchmarking the time performance of each format technology solution and selecting the most effective ones. The research about suitable algorithms for the customer's domain is probably unnecessary as it has already been made in Martina Fuskova's thesis.

8.1.2 Add support for complex types of anomalies

Also, the room for the code extension is the logic recognizing more sophisticated types of anomalies, not just the point anomalies.

This amendment would not require much effort, as the fact that there exist several types of anomalies – not just point anomalies – was taken into account when designing the framework.

However, the valid question is whether this innovation is worth implementing because some more complex types of anomalies can be inferred from point anomalies. It would be necessary to research algorithms dedicated to detecting such type of aberrations and decide what would be the best solution.

8.1.3 Add support for pre- and post-processing of data

Finally, the very exciting might be to analyze possibilities of pre- and post-processing of data. Again, the best solution would be to have all logic stored in a file and then just load it and execute it in a Java run-time. From this point of view, the most auspicious is the PFA technology. A solution could be to chain the PFA script with some anomaly detection model.

Bibliography

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[2] Issam El Naqa and Martin J Murphy. "What is machine learning?" In: *machine learning in radiation oncology*. Springer, 2015, pp. 3–11.

[3] Unknown. *Supervised Learning*. IBM Cloud Education. Aug. 19, 2020. URL: https://www.ibm.com/cloud/learn/supervised-learning (visited on 05/02/2021).

[4] Diogo M. Camacho et al. "Next-Generation Machine Learning for Biological Networks". In: *Cell* 173.7 (2018), pp. 1581–1592. ISSN: 0092-8674. DOI: https://doi.org/10.1016/j.cell.2018.05.015. URL: https://www.sciencedirect.com/science/article/pii/S0092867418305920.

[5] James Chen. *Instrument*. Investopedia. Aug. 15, 2019. URL: https://www.investopedia.com/terms/i/instrument.asp (visited on 04/19/2021).

[6] Tim Smith and Gordon Scott. *Broker*. Investopedia. Oct. 6, 2020. URL: https://www.investopedia.com/terms/b/broker.asp (visited on 04/19/2021).

[7] Gordon Scott. *Financial Information Exchange (FIX)*. Investopedia. Mar. 28, 2021. URL: https://www.investopedia.com/terms/f/financial-information-exchange.asp (visited on 04/19/2021).

[8] Kosta Zenelis et al. Documentation. FIX Trading Community. URL: https://fiximate.fixtrading.org/ (visited on 04/19/2021).

[9] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. "Generic and scalable framework for automated time-series anomaly detection". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1939–1947.

[10] Felipe Pezoa et al. "Foundations of JSON schema". In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 263–273.

[11] Mauricio Arango and Alex Ardel. "Machine Learning on Streaming Data via Integration of Oracle Advanced Analytics and Oracle Stream Explorer". In: (2016).

[12] Mark Hornick. *Oracle Machine Learning Product Management*. online. 2020. URL: https://www.oracle.com/a/tech/docs/oracle-machine-learning-overview-and-roadmap.pdf (visited on 04/08/2021).

[13] Álvaro López García.
"DEEPaaS API: a REST API for Machine Learning and Deep Learning models".
In: *Journal of Open Source Software* 4.42 (Oct. 25, 2019), p. 1517. ISSN: 2475-9066.
DOI: 10.21105/joss.01517. URL: http://dx.doi.org/10.21105/joss.01517.

[14] Álvaro López García.
DEEPaaS API: a REST API for Machine Learning and Deep Learning models.
Towards Data Science. Apr. 3, 2020. URL: https://towardsdatascience.com/publish-your-models-with-a-standard-rest-api-94de1e1da682 (visited on 04/09/2021).

[15] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python".
In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[16] R.A.Fisher. *iris*. 2018. DOI: 10.21227/rz7n-kj20.
URL: https://dx.doi.org/10.21227/rz7n-kj20.

[17] R. Grossman et al. "The management and mining of multiple predictive models using the predictive modeling markup language".
In: *Information and Software Technology* 41.9 (1999), pp. 589–595. ISSN: 0950-5849.
DOI: https://doi.org/10.1016/S0950-5849(99)00022-1.
URL: https://www.sciencedirect.com/science/article/pii/S0950584999000221.

[18] Unknown. *PMML 4.4.1 - General Structure*. Data Mining Group.
URL: http://dmg.org/pmml/v4-4-1/GeneralStructure.html (visited on 04/09/2021).

[19] Bray T. Paoli J. Sperberg-McQueen C. et al. *Extensible Markup Language (XML) 1.0.*
Feb. 2008. URL: http://www.w3.org/XML (visited on 02/05/2015).

[20] Unknown. *PMML 4.4.1 - Nearest Neighbors*. Data Mining Group.
URL: http://dmg.org/pmml/v4-4-1/KNN.html#xsdElement_NearestNeighborModel
(visited on 04/09/2021).

[21] Unknown. *PMML 4.4.1 - Anomaly Detection Models*. Data Mining Group. URL:
http://dmg.org/pmml/v4-4-1/AnomalyDetectionModel.html (visited on 04/09/2021).

[22] Unknown. *Data Mining Group*. Data Mining Group.
URL: http://dmg.org/ (visited on 04/11/2021).

[23] Unknown. *Format of a PFA document*. Data Mining Group.
URL: http://dmg.org/pfa/docs/document_structure/ (visited on 04/11/2021).

[24] Jim Pivarski, Collin Bennett, and Robert L Grossman.
"Deploying analytics with the portable format for analytics (PFA)".
In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 579–588.

[25] Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2015.

[26] Sean T Allen, Matthew Jankowski, and Peter Pathirana.
Storm Applied: Strategies for real-time event processing. Manning Publications Co., 2015.

[27] Eric Boyd. *Microsoft and Facebook create open ecosystem for AI model interoperability.*
Microsoft Corporation. Sept. 7, 2017.
URL: https://azure.microsoft.com/en-us/blog/microsoft-and-facebook-create-open-ecosystem-for-ai-model-interoperability/ (visited on 04/11/2021).

[28] Unknown. *Technical Design*. ONNX.
URL: https://onnx.ai/about.html (visited on 04/14/2021).

[29] Junjie Bai et al. Open Neural Network Exchange - ONNX. URL:
https://github.com/onnx/onnx/blob/master/docs/IR.md (visited on 04/14/2021).

[30] Unknown. *tf.keras.Model*. TensorFlow.
URL: https://www.tensorflow.org/api_docs/python/tf/keras/Model (visited on 04/29/2021).

[31] Daniel Ellis. *A Tour of SavedModel Signatures*. Tensorflow. Mar. 2021. URL: https://blog.tensorflow.org/2021/03/a-tour-of-savedmodel-signatures.html (visited on 04/14/2021).

[32] *Openscoring.io*. Openscoring Ltd. URL: https://openscoring.io/ (visited on 05/07/2021).

[33] Villu Ruusmann. *Java libraries for producing and consuming PMML documents*. Openscoring Ltd. URL: https://github.com/jpmml (visited on 04/15/2021).

[34] Villu Ruusmann. *JPMML-Evaluator*. Openscoring Ltd. URL: https://github.com/jpmml/jpmml-evaluator (visited on 04/15/2021).

[35] Steven M. Mortimer et al. *Hadrian: implementations of the PFA specification*. ModelOp. URL: https://github.com/modelop/hadrian/wiki/PrettyPFA-Simple-Example (visited on 04/15/2021).

[36] Ankit Mahato. *Titus2*. Version 1.2.0. URL: https://pypi.org/project/titus2/.

[37] Faith Xu. Youtube. Dec. 4, 2018. URL: https://www.youtube.com/watch?v=qy7X2JGLUC4 (visited on 04/15/2021).

[38] Guoyu Wang. *ONNX Runtime Java API*. Documentation. Microsoft Corporation. URL: https://www.onnxruntime.ai/docs/reference/api/java-api.html (visited on 04/16/2021).

[39] Unknown. *Install TensorFlow Java*. Documentation. TensorFlow. URL: https://www.tensorflow.org/jvm/install (visited on 04/17/2021).

[40] *IntelliJ IDEA*. Version Community addition 2020.2. Jet Brains. URL: https://www.jetbrains.com/idea/ (visited on 04/23/2021).

[41] Unknown. *Class diagram toolbar, context menu, and legend*. Documentation. Jet Brains. Mar. 8, 2021. URL: https://www.jetbrains.com/help/idea/class-diagram-toolbar-and-context-menu.html (visited on 04/23/2021).

[42] Bran Selic et al. *OMG Unified Modeling Language (Version 2.5)*. Mar. 2015. URL: https://www.researchgate.net/publication/281633784_OMG_Unified_Modeling_Language_Version_25 (visited on 04/23/2021).

[43] Stephen Colebourne and Oracle. *LocalDateTime*. Documentation. URL: https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html (visited on 04/23/2021).

[44] Google. *Gson*. Version 2.8.6. URL: https://github.com/google/gson (visited on 04/23/2021).

[45] Doug Lea. *Interface Callable¡V¿*. Documentation. Oracle Systems Corporation. URL: https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/package-summary.html (visited on 04/23/2021).

[46] Doug Lea. *Interface Future¡V¿*. Documentation. Oracle Systems Corporation. URL: https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/package-summary.html (visited on 04/23/2021).

[47] Kent Beck et al. *Junit*. Version 4.12. URL: https://github.com/junit-team/junit4 (visited on 04/29/2021).

[48] Unknown. *Javadoc*. Oracle Systems Corporation. URL: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html (visited on 04/29/2021).

[49] Miroslav Čepek, Kamil Dedecius, and Karel Klouda. *BIE-VZD lecture 5*. Czech Technical University in Prague, Faculty of Information Technology. Oct. 23, 2020. URL: https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/05/BI-VZD-05-en-slides.pdf (visited on 04/30/2021).

[50] Unknown. *Recurrent Neural Networks*. IBM Cloud Education. Sept. 14, 2020. URL: https://www.ibm.com/cloud/learn/recurrent-neural-networks (visited on 04/30/2021).

[51] Jeremy Jordan. *Introduction to autoencoders*. Mar. 19, 2018. URL: https://www.jeremyjordan.me/autoencoders/#:~:text=Autoencoders%5C%20are%5C%20an%5C%20unsupervised%5C%20learning,representation%5C%20of%5C%20the%5C%20original%5C%20input. (visited on 05/01/2021).

[52] ladylittlebee. *LinReg,KNN,SVR,DecisionTreeRandomForest,TimeSeries*. 2019. URL: https://www.kaggle.com/ladylittlebee/linreg-knn-svr-decisiontreerandomforest-timeseries/notebook (visited on 04/24/2021).

[53] Villu Ruusmann. *Java libraries for producing and consuming PMML documents*. Openscoring Ltd. URL: https://github.com/jpmml/sklearn2pmml (visited on 04/15/2021).

[54] Venelin Valkov. *Time Series Anomaly Detection with LSTM Autoencoders using Keras in Python*. Curiously. Nov. 24, 2019. URL: https://curiousily.com/posts/anomaly-detection-in-time-series-with-lstms-using-keras-in-python/ (visited on 05/01/2021).

[55] Venelin Valkov. *Time Series Anomaly Detection with LSTM Autoencoders using Keras in Python*. Youtube. Dec. 29, 2019. URL: https://www.youtube.com/watch?v=H4J74KstHTE (visited on 05/01/2021).

[56] Will Kenton and Michael J Boyle. *S&P 500 Index – Standard & Poor's 500 Index*. Investopedia. Mar. 23, 2021. URL: https://www.investopedia.com/terms/s/sp500.asp (visited on 04/29/2021).

[57] Martina Fusková. *Anomaly detection for stock market trading data*. Bachelor thesis, Supervisor Kofroň, Jan. Faculty of Mathematics, Physics, Department of Distributed, and Dependable Systems. 2020. URL: https://dspace.cuni.cz/handle/20.500.11956/120934 (visited on 04/27/2021).

Contents of the enclosed media

- `readme.txt` ..... a brief description of the content
- `src`
  - `impl` ..... the source code of the implementation
  - `doc` ..... the Javadoc documentation of the source code
- `thesis` ..... the source folder for LaTeX
- `text` ..... the text of the thesis
  - `thesis.pdf` ..... the text of the thesis in PDF format