Jussi Kolehmainen

Predicting Complex Events in Sensor Data

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 08.08.2013

Thesis supervisor:

Prof. Ahti Salo

Thesis advisor:

M.Sc. (Tech.) Antti Aalto

Aalto University
School of Science

AALTO-YLIOPISTO PERUSTIETEIDEN KORKEAKOULU	DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Jussi Kolehmainen

Työn nimi: Monimutkaisten tapahtumien ennustaminen sensoridatasta

Päivämäärä: 08.08.2013 Kieli: Englanti Sivumäärä:7+74

Matematiikan ja systeemianalyysin laitos

Professuuri: Systeemi- ja operaatiotutkimus Koodi: T3020

Valvoja: Prof. Ahti Salo

Ohjaaja: DI Antti Aalto

Monimutkaisten tapahtumien prosessointi (CEP) on teknologia, joka analysoi datavirtoja ja tunnistaa monimutkaisia tilanteita reaaliajassa. Tunnistettavat tilanteet määritellään EPL-lauseilla, joiden kirjoittamiseen tarvitaan tutkittavaa ilmiötä tuntevia asiantuntijoita. Yhdessä matemaattisten ennustemenetelmien kanssa CEP mahdollistaa tapahtumien ennustamisen ja niistä varoittamisen.

Tässä työssä käyn läpi mitä ennustava tapahtumaprosessointi on ja kuinka eri komponenteista voidaan kasata järjestelmä reaalimaailman tapahtumien ennustamiseen. Matemaattisina työkaluina käytän kahta mallia: etäisyysmittaan perustuvaa mallia ja piirrevektoriin perustuvaa mallia. Ensimmäinen käyttää DTW:tä etäisyysmittana ja k:n lähimmän naapurin (kNN) algoritmia luokitteluun. Jälkimmäinen pohjautuu wavelet-analysiin ja tukivektorikoneisiin (SVM).

Kokeellisessä osiossa esittelen taloautomaation sovellutuksena ennustavalle tapahtumaprosessoinnille. Testeissä käytän todellisia sensoreita, joille ennustan sisäilmayhdistyksen asettamien raja-arvojen ylityksiä. Tämän lisäksi arvioin rakentamani systeemin suorituskykyä.

Hiilidioksidille ja haihtuville orgaanisille yhdisteille ensimmäinen malli antaa oikeita hälytyksiä yli 75 %:ssa tapauksista ja pääsee alle 10 %:iin väärien hälytysten osalta. Jälkimmäinen malli toimii nopeammin, mutta sen konfigurointi osoittautuu haastavaksi, minkä takia tulokset ovat huonompia. Jatkotutkimusten aiheiksi suosittelen erityisesti järkevämpää tapahtumamäärittelyä ja systeemin aikaparametrien parempaa optimointia.

Avainsanat: monimutkaisten tapahtumien prosessointi, ennustaminen, ympäristödata, taloautomaatio, tukivektorikoneet, wavelet-analyysi

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Jussi Kolehmainen		
Title: Predicting Complex Events in Sensor Data		
Date: 08.08.2013	Language: English	Number of pages:7+74
Department of Mathematics and System Analysis		
Professorship: Systems and Operations Research		Code: T3020
Supervisor: Prof. Ahti Salo		
Advisor: M.Sc. (Tech.) Antti Aalto		

Complex event processing (CEP) analyzes data streams and detects complicated situations in real-time. Domain experts write effective EPL (event processing language) queries to define complex events that are detected. In combination with predictive analytics (PA), which uses mathematical models to predict the future, a framework for predicting complex events can be designed.

In this thesis I describe how predictive event processing works and how a proof-of-concept framework can be built. As prediction tools I use two models: a distance-based model and a feature-based model. The former uses dynamic time warping (DTW) and k-nearest neighbour (kNN) algorithm while the latter employs wavelet analysis and support vector machines (SVMs).

As an application of predictive complex event processing I consider house automation and present a real-life case study for the experimental section. The goal is to predict when a certain variable exceeds a limit value for a certain period of time. I also evaluate the performance of the system.

With two variables, CO_2 and VOC (volatile organic compounds), the first, distance-based model performs better with correct alarm rate of over 75 % and false alarm rate of under 10 %. The second, feature-based model turns out to be faster but more difficult to configure properly. More meaningful complex events and more thorough time parameter optimization are suggested for future research.

Keywords: complex event processing, predictive analytics, environmental data, house automation, support vector machines, wavelet analysis

Acknowledgements

First, I want to express my deepest gratitude towards my family who has always supported and encoraged me in my studies. You have helped me to achieve yet another goal in my life. Second, I want to thank my supervisor, Ahti Salo, and my instructor, Antti Aalto, for guiding me through the writing process. Third, I want to thank my employer HiQ Finland and especially Jukka-Petri Sahlberg for permitting me to write this thesis for the MMEA research program. I also want to thank the research group of environmental informatics at the University of Eastern Finland and especially Mauno Rönkkö for giving me access to their test data. All in all, thanks to everyone who supported me and helped me through this long but rewarding process.

Albert Einstein once said: *"Once we accept our limits, we go beyond them"*. This thesis used to be one of them. Now I have to accept that my limits have once again moved a big step further.

Otaniemi, 08.08.2013

Jussi Kolehmainen

Contents

Symbols and abbreviations

Symbols

$\mathbf{AC_d}$	ROC distance from optimal classifier
C	SVM soft margin parameter
γ	SVM kernel parameter
k	kNN parameter
radius	DTW warping path parameter

Abbreviations

DTW	Dynamic Time Warping
DWT	Discrete Wavelet Transform
EPA	Event Processing Agent
EPN	Complex Event Network
CEP	Complex Event Processing
kNN	k-Nearest Neighbor
PEPN	Predictive Event Processing Network
ROC	Receiving Operator Characteristics
SVM	Support Vector Machine
P	Positives, number positive instances
N	Negatives, number of negative instances
TP	True Positives
FP	False Positives
TN	True Negatives
FN	False Negatives
TPR	True Positive Rate
FPR	False Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate

Chapter 1

Introduction

1.1 Background

As the size of the digital universe, that is, the total size of digital data in the world is approaching zettabytes (trillion gigabytes) [22], more and more ways are needed to make sense of that information. One of the biggest technology trends in this decade, big data refers to the analysis of large volumes of data. The definition of the term big data varies a lot but some common characteristics can usually be found: conventional database systems are not enough because the data is too big, moves too fast or does not fit in the current data structures. [13]

Complex event processing (CEP) can be thought as a branch of big data in which the speed of the data flow is too much for the ordinary databases to handle. CEP tackles this issue by providing event-driven processing, for example pattern detection and causality analysis, with extremely low latencies. [17] It reverses the idea of conventional databases: instead of running queries against an existing data set, CEP uses predefined queries against which the data is run.

Despite these evident advantages of CEP in real-time data processing, it is only a technological opportunity without domain expertise. Like a database expert writes complex SQL queries that return the desired dataset, a CEP expert writes EPL queries that filter the data flow in some way. However, it is possible to reduce the amount of human work needed by extending a CEP engine with other advanced technologies, such as machine learning (ML) and predictive analytics (PA). [21] In this thesis, I will present a framework for predictive complex event processing (PCEP) that is capable of predicting an event beforehand by learning the pattern that precedes the event.

A promising field of application for complex event processing is ambient intelligence, or house automation, which comprises smart homes that are able to change the living environment depending on various indicators. [3] Smart homes gather information, for instance, about the whereabouts of the people living in them, air quality and electricity consumption. From these factors the smart homes can then adjust heating, ventilation and lights accordingly. CEP can provide an efficient platform for this real-time data gathering and analysis. This thesis contains a real-world study case from the field of house automation. By combining complex event processing, predictive analytics, and sensor data, smart homes can be made to think one step ahead.

The learning and prediction phase are implemented with relatively new techniques, namely Wavelet Analysis and Support Vector Machines (SVMs). While Fourier Transform captures only the frequency domain properties of a signal, wavelets are localized in both time and frequency domains. [20] Thus, wavelets can be used to extract informative patterns from sensor data. These patterns, then, are classified with binary classifiers, SVMs, into patterns that precede certain events and patterns that do not.

As a comparative method a k-Nearest Neighbor algorithm in combination with Dynamic Time Warping (DTW) measure is used. DTW is an extension of Euclidean Distance and it is more suitable for comparing time series.

This thesis is part of a five-year research program called The Measurement, Monitoring and Environmental Assessment (MMEA). The program aims at creating new tools for environmental data usage in both consumer and industry sector. The environmental data available from sensor data producers will be made available to data consumers in an open-source marketplace where the parties can connect with their own software. [10]

1.2 Motivation for Predictive Event Processing

A system that is capable of predicting events instead of just detecting them can prevent undesirable conditions from happening. An apparent field of application where these undesirable conditions might be harmful for humans is house automation. Moreover, a system like this can save energy by adjusting control variables more efficiently and provide the residents automated changes of the living conditions. [52]

A good example of harmful living conditions is the rise in the concentration of carbon monoxide, which is a highly toxic gas. Being completely colorless, odorless and tasteless, it can only be detected with specific sensors. However, detecting the risen concentration is not enough as the poisonous gas is already present and the residents are in danger. By detecting the increase the house could alert the residents and set the ventilation to full-speed. [65] Another example of harmful situation that could be avoided with a predictive system is flood. The rise in the water level that leads to a flood must be distinguished from the normal seasonal variation. [58]

An intelligent control of heating, especially floor heating, can create significant energy savings because of the long delay between controlling the heating settings and the actual heating effects. In a classical heating system, such as a radiator, the delay does not need to be taken into account because it reacts to the control in minutes and emits its energy to the surroundings. An intelligent floor heating, however, requires predicting the future energy prices and outside temperatures, because the floor should be heated during off-peak hours when the prices are lower or when outside temperatures are about to fall. [8]

Comfortable living conditions are not achieved by merely avoiding harmful conditions. A constant temperature, which increases living convenience remarkably, cannot be maintained if the heating system does not adapt to substantial changes in outside temperature beforehand. Also the concentration of volatile organic compounds (VOC), which is one of the test cases in this thesis, should be kept as low as possible in order to maintain good air quality. Again, a predictive component with machine learning algorithms is needed to adjust the ventilation before the concentration hits a critical level. The system should also adapt to the residents' lifestyle and schedule the changes in indoor air conditions accordingly. [52]

In addition to house automation, predictive event processing can be used in such fields as financial markets and traffic control. As the stock market is nowadays mostly run by automated algorithms, the speed of data processing is a critical factor. Combined with predictive machine learning components, such system could attain crucial advantage against its competitors. Traffic control system, in turn, could predict traffic jams from past data. By varying speed limits and guiding vehicles to less occupied roads, it could reduce emissions and travel times. [4]

1.3 Research Questions and The Scope of This Thesis

Complex Event Processing (CEP) is mostly used for detecting known patterns in the data stream. When CEP is used to analyze our surrounding world, it is relatively easy to detect a certain situation that can be undesirable for humans, animals or some materials. However, the actions to prevent those conditions from happening should usually be taken long before that as it was mentioned in the previous section.

The main research question of this thesis is to formulate a framework for combining CEP and Predictive Analytics (PA). There is plenty of research done from both areas but not many papers have dealt with combining them. The most important inspiration for this thesis is the work done by Hungarian scientist Lajos Jeno Fülöp and his colleagues at Nokia Siemens Network. They tried to detect and predict the number of persons entering and leaving a building. Their application employed decision tree algorithms, such as BFTree, for detecting conditions that precede a traffic peak. [21]

One of the novel topics in this thesis is the data handling with CEP. The framework not only detects the complex events with CEP but also creates sliding windows for collecting the data training and labeling it into positive and negative samples. The goal of this approach is to test the capabilities of CEP for more than just its main purpose, detection of events that have already been occurred.

The two models used in this thesis were chosen by suggestions from research papers and literature. The first model, which uses DTW and kNN, is a very simple approach for time series classification. DTW has been used in many research applications and has been proven to achieve one of the smallest error rates with many data sets (e.g. [32]).

The combination of Wavelets and SVMs were chosen as our second model because it represents a much more modern approach. Even though Haar Wavelets were first discovered in the early 1900s, the theory of Wavelets has been evolving still in the 2000s in the form of image compression and trend detection [60]. The Support Vector Machines (SVMs) were first mentioned in 1979, but formally introduced by Vladimig N. Vapnik as late as in 1995 [5].

In the light of past results with DTW and SVM based classification it would be a big surprise if the latter would achieve better performance. [32] [29] The main reason for this is that tuning of both Wavelets and SVMs is much more challenging task than using simple DTW and kNN based model which has only two parameters.

1.4 Structure of This Thesis

Chapters 2.1 and 2.2 present the principles of complex event processing (CEP). Chapter 2.3 introduces a CEP Engine called Esper which is used in the experimental part of this thesis. The Event Processing Language (EPL) for Esper is also reviewed. Chapter 2.4 briefly reviews some applications of CEP.

Chapter 3.1 defines predictive analytics (PA) and Chapter 3.2 discusses how it can be integrated with CEP. Chapter 3.3 formulates the problem for time series prediction. Chapters 3.4 and 3.5 present mathematical models for time series prediction. The former focuses on distance based methods while the latter describes what feature based classification is. Chapter 3.6 shows how the performance of these methods can be evaluated.

Chapter 4 begins with introducing the case study and the available test data. Then, Chapter 4.2 continues with describing the implementation of a Predictive Event Processing Network (PEPN). Chapter 4.3 briefly reviews the MMEA platform architecture and explains how the predictive component could be integrated. Chapter 4.4 defines the complex events we are trying to predict and presents methods for parameter tuning.

Chapter 5 presents the performance results of the classifiers. Then, it discusses computational performance.

Chapter 6 sums up the results of this thesis and discusses further research possibilities from the field of predictive complex event processing.

Chapter 2

Complex Event Processing

2.1 Events and Patterns

One definition for an event is "anything that happens, or is contemplated as happening". [37] According to this wide definition, there is a huge number of events occurring around us all the time. Examples of these events include opening a door, receiving a measurement value from a sensor and the ending of the World War II. It can easily be seen that we need to further classify events in order to be able to define a system for event processing.

Events can be divided into two groups: simple and complex events. [14] A simple event is any single instance that comes into our system from sensors. Thus, a simple event is a single measurement value. Complex events, on the other hand, are derived from the single events and are something that we want to detect from the event stream. They are the result of the event processing system. Examples of complex events that we are interested in thesis are a single measurement variable rising at a certain speed and a group of variables exceeding a threshold value for a certain amount of time.

An event sequence is "a time-ordered sequence of events". [62] For example, a series of consecutive measurement values form an event sequence. An event pattern is an event sequence with given values for the events. [37] Hence, a peak and a rise in a variable are both event sequences but are classified as different event patterns. In this thesis I assume that the complex events of our interest can be predicted by detecting the event pattern that precedes the event.

2.2 Overview of a General Event Processing System

An outline of a complex event processing system is presented in Figure 2.1. The inputs of the network are the event sources that produce simple events. They can be, for instance, thermometers or movement sensors. An Event Processing Agent (EPA) is a node in the network that consumes events, performs some predefined tasks, and produces new events based on its rules and input events. Possible tasks for an EPA include event filtering, aggregating and pattern detection. [21]

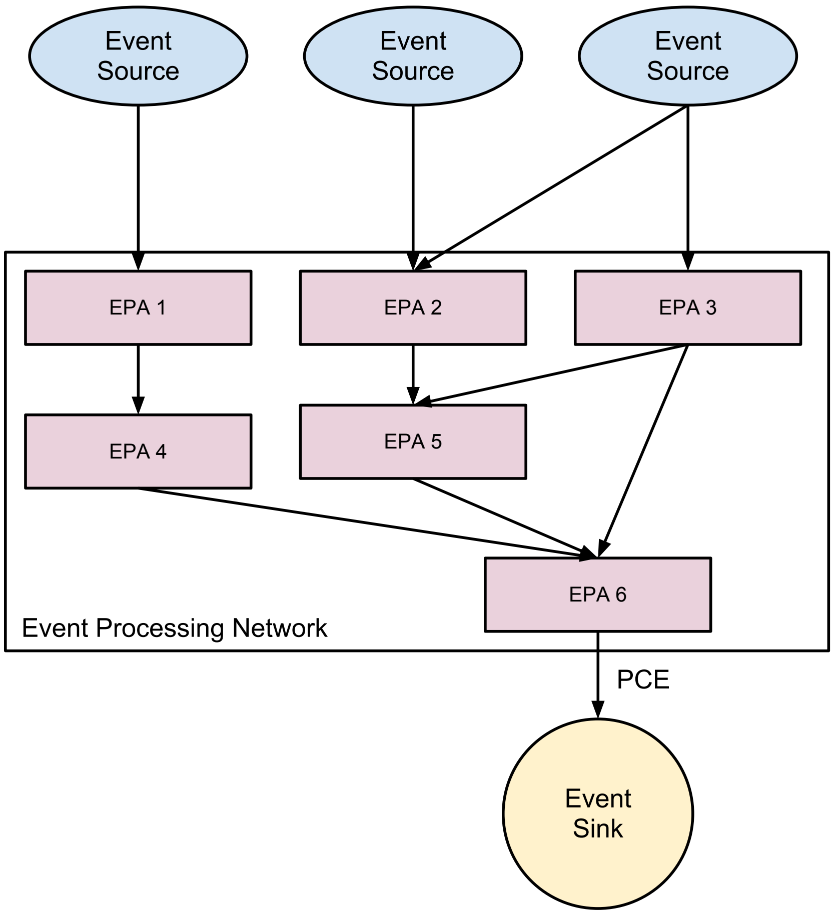

Figure 2.1: Event Processing Network.

All the nodes in the network are connected via event channels (arrows in the figure). It is suggested that at this point the event channels are high-level abstractions and we do not limit the types they can handle nor the number of sources or consumers attached to them. [14]

A group of interconnected EPAs form an Event Processing Network (EPN). An EPN has the following properties: it can by dynamic (EPAs can be created and destroyed), it may contain feedback loops and it can be distributed across multiple physical computers. [37] However, in the study case of this thesis we do not need any of those properties because the complex events we are interested in are rather simple ones.

As a result, an EPN produces a Primary Complex Event (PCE) that goes to the event sink. A PCE has the prefix primary in order to distinguish it from the complex events passed between EPAs. A PCE is the event whose future occurrences we are trying to predict in this thesis. The event sink represents all the parties who are interested in the complex event in question, including both automated machines that are triggered to perform some action and human workers who are notified when something interesting happens. [21]

2.3 CEP Engines and Processing Languages

There is a wide range of Complex Event Processing Engines with different properties available on the market. First started in the Bell Labs in the mid 90's, the CEP business sector has spanned into more than 20 widely used products from all the major IT houses, such as IBM, Microsoft and Oracle. [59]

CEP engines can be divided into four subcategories based on abstraction type for event detection and action triggering. In the first category, Data Stream Query Languages, a SQL-like language is used to create relational queries against the data stream. In the second category, Production Rule Languages, queries are based on a set of condition-action pairs stored in the working memory. When a condition becomes true, the corresponding actions is fired. In the third category, Composition-Operator-Based Languages, operators are used to define complex causal queries based on simple queries, such as whether or not a certain event happens. The last category is rarer; the languages in it use logical XML formulas for automated reasoning in a Semantic Web. [24]

Most CEP engines have several common features that are used to handle the data stream. As all the events in CEP are somehow typed, filtering can be used to select only the ones that we are interested in. Windows, specified either with time interval or a number of events, allow us to use a subset of the stream for processing. Data aggregation, conjunctions and disjunctions can be used to create more abstract levels of events. Temporal and causal relationships between events are efficient in reasoning complex sequences and patterns of events. Of course, negation and counting methods are available for detecting the presence or absence of events. [38]

In MMEA program, and consequently in this thesis, we use an open-source, Java-based CEP engine called Esper. Esper has properties from the first three categories listed above. It uses Event Processing Language (EPL) to specify expressions for pattern matching and for detecting the presence or absence of events [16]. A standard EPL query is of the form

```
INSERT INTO
        [stream]
SELECT
        [attributes]
FROM
        [stream]
WHERE
        [condition]
GROUP BY
        [attribute]
HAVING
        [condition]
```

The *SELECT* clause is used to select which attributes we want to catch from each event that matches our filter. The optional *INSERT INTO* clause puts these attributes into a new stream. The *FROM* clause specifies the streams and corresponding time windows we are looking into. The *WHERE* clause can filter the events, for example, by giving a threshold value for a certain event attribute. The *GROUP BY* clause is used to aggregate several events of the same type into an abstraction event, for instance, by event type. The *HAVING* clause is used filter the aggregated events formed by *GROUP BY*. [24]

In Esper, Events can be POJOs (Plain Old Java Objects), Maps (key-value-dictionaries) or XML documents. All event types have attributes that can further have sub-attributes of any type. [16] In the following, I will present some examples of Java POJOs in Esper.

Say we have the following Java class

```
public class Sensor {
        int id;
        String name;
        Sensor[] groupSensors;
}
```

A Sensor has an identification number, name and a list of other sensor belonging to the same group. Furthermore, say we have an event type called *SensorEvent* which represents a single measurement that comes from a sensor:

```
public class SensorEvent {
        Sensor sensor;
        Timestamp timestamp;
        Map<String, Double> measurements;
}
```

A *SensorEvent* has a reference to the sensor from which the event originated, a timestamp that captures the exact point in time when the measurement was made and a map that contains key-value pairs where the key is the variable name (e.g. temperature) and the value is the measurement value (e.g. 10.9 degrees celsius).

Now we can refer to the event attributes in an EPL query in the following way:

Simple: SensorEvent.timestamp
measurement time

Nested: SensorEvent.sensor.id
sensor id

Indexed: SensorEvent.sensor.groupSensors[0].name
name of the first sensor in the same group

Mapped: SensorEvent.measurements('temperature')
value of the temperature measurement

Next, I will present some examples of the query types we might need in the study case of this thesis. To select the name of the sensors that have the temperature exceeding 20.0 degrees can be done using the following query:

```
SELECT
        sensor.name
FROM
        SensorEvent
WHERE
        measurements('temperature') > 20.0
```

To retrieve the average humidity from each sensor every 30 minutes we can use a timed window:

```
SELECT
        avg(measurements('humidity'))
FROM
        SensorEvent.win:time_batch(30 min)
GROUP BY
        sensor.id
```

To define a sliding window of 30 minutes that is triggered on every new instance, we can change the *win:time_batch* to just *win:time*. A time interval can be defined with *timer:interval*. Combined with the patterns described below, it is useful, for example, when a certain event has to happen within 10 minutes of another event.

As an example of event pattern detection, let's assume we have a stream (called Stream) of events: A_1 A_2 B_1. This example was originally shown in Esper Reference [16]. Now the pattern

```
SELECT
        every a=A -> b=B
FROM
        Stream
```

matches each A followed by B ([A_1 B_1], [A_2 B_1]) but the pattern

```
SELECT
        every a=A -> (b=B and not A)
FROM
        Stream
```

matches every A followed by B but not A. In this case, the resulting event is only [A_2 B_1].

As one can see, by combining windows, relational clauses and SQL-like procedures, such as sub-queries and joins, we are able to define quite complex queries that would be hard to implement in conventional programming methods. This thesis does not focus much on the writing of effective EPL queries for detecting complex events, but rather assumes that the queries have already been defined by some domain expert.

By default, Esper uses system's time when creating windows and handling causality. This behavior can be bypassed by turning Esper's internal clock off and constantly sending *CurrentTimeEvent* objects into the engine with a selected timestamp. [16]

2.4 Applications of CEP

Complex event processing (CEP) has found applications in many fields from Business Process Monitoring (BPM) to intrusion detection in computer networks. In this chapter I present some successful examples of using CEP for high-speed data processing.

Radio frequency identification (RFID) is a technology for identifying physical objects by equipping them with a remote-readable tags that can be read using radio-frequency electromagnetic fields. Then, the lifespan of the object can be monitored more accurately than with any other method. In big industrial companies this produces a huge amount of data that has to processed and analyzed in real-time in order to get the maximum benefit from the RFID technology. CEP provides a great platform for filtering the RFID data, aggregating semantic information into the data and routing the data to customers and suppliers. [61]

In the financial sector, where tremendous amounts of data are passed between banks and customers constantly, it is no surprise that the data can be transferred and processed effectively with CEP. Example use cases are detecting suspicious money transfers and credit card frauds by monitoring where the credit card has been used and how much money has been withdrawn with it. In stock market CEP can be used to follow market trends and correlations between different stocks. [2]

Business process management (BPM) and CEP can be integrated to provide valuable information to customer relationship management (CRM). By monitoring real-time customer interactions, such as clicks in an e-commerce or comments on a blog, business providers can create automated actions that personalize service to each customer. Event-driven actions can even indicate the customer's emotional state, which may be used for targeted advertising. [33]

In combination with predictive analytics (PA), which we will discuss more in chapter 3, CEP can be utilized to detect and locate failures in component based systems by online tracking. Chen et al. [9] monitored the distribution between signal and noise subspaces, and detected failures with online distribution learning algorithms.

Chapter 3

Predictive Analytics

3.1 General

Predictive analytics (PA) is a broad field of applied mathematics. First, it includes all the statistical models and empirical methods that are used to create empirical predictions. Second, also predictive power, that is, methods for assessing the quality of predictions is a part of PA. [51] Another purpose of PA is to guide theory building, theory testing and relevance assessment. [11]

One way to classify PA methods is to divide them into predictive models, descriptive models and decision models. Predictive models look for the most significant explanatory variables with which they are able to predict the dependent variables. Descriptive models, in turn, try to find as many relationships as possible, leading to segmented models that describe the world as it is. Decision models use optimization techniques to find the most optimal decision based on possible outcomes. [19]

Another way to classify PA methods is to distinguish between methods that predict the present and methods that shape the future. Predicting the present means applying patterns of current behavior with as much historic data as possible. These methods use classification, regression, clustering and all data mining techniques to predict similar occurrences in the future. Shaping the future has to do with generating new standards after the underlying assumptions have changed. While predicting the present can be thought as fitting a curve into the existing data, shaping the future creates a totally new curve with new assumptions and anticipated behaviors. [6] In this thesis, all the used methods can be classified as predicting the present.

The methods of PA can also be divided into regression and machine learning techniques. The simplest model is probably linear regression, which tries to find a linear model between a set of independent variables and a dependent variable by minimizing the squared error [40]. Time series models assume that the process we are investigating has some kind of internal structure with autocorrelation, trend or seasonal variation. This structure may be discovered with either time-domain or frequency-domain analysis. [27] The former is based on auto-correlation and cross-correlation analysis, while the latter is carried out with spectral or wavelet analysis, which I will talk about more in chapter 3.5.

Machine learning algorithms form a model, based on clustering or historic data, that is able to predict the class of the dependent value without necessarily knowing the exact mathematical structure behind it. As an example of clustering based methods, unsupervised Artificial Neural Networks (ANNs) are biology-inspired complex networks that evolve into a form that can classify new instances into a group with the most similar instances. [39] Models based on historic data are called supervised models and they can, for example, find a classifier function that separates the training data most effectively. I will talk more about classifiers in chapter 3.5 when I introduce Support Vector Machines.

3.2 Combining CEP and PA

The work of Fulop et al. [21] lists several synergies and differences of CEP and PA which are briefly summarized in this section.

Complex event processing (CEP) and Predictive Analytics (PA) are similar in the sense that they try to somehow make sense of large datasets, either in real-time or from historic data. The biggest differences between CEP and PA stem from the timing of the reasoning process. While CEP processes data real-time and detects the events only after they have occurred, PA, as the name suggests, tries to predict events by detecting the patterns that lead to them.

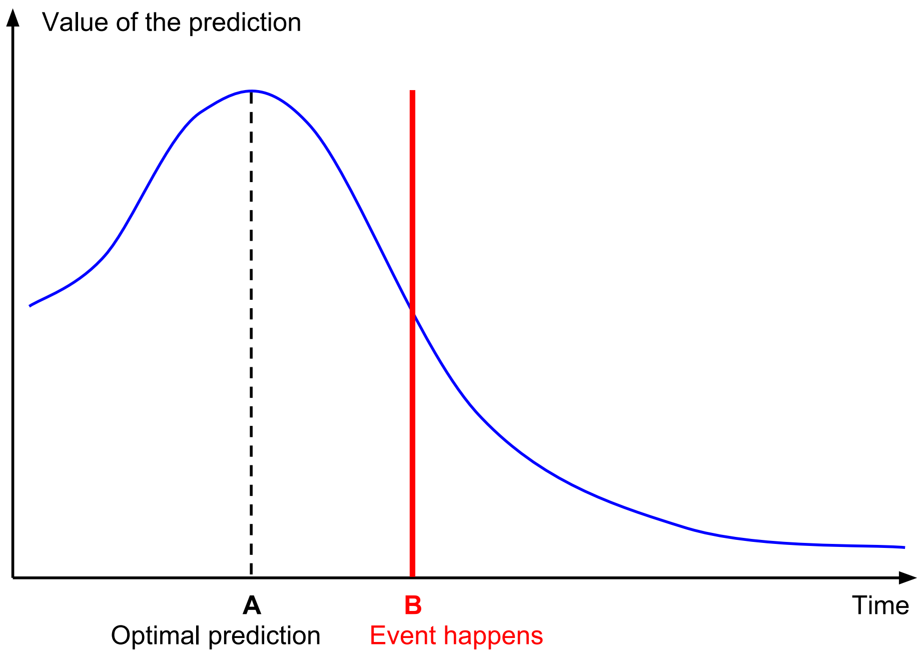

Figure 3.1: Expected value of event detection as a function of time. An optimal prediction is made at the point A. The event happens at the point B, after which detecting the event is still valuable.

The (expected) value of detecting an event depends directly on time, which is illustrated in Figure 3.1. Up to point A, value increases with increased accuracy. This is because long time predictions are not usually accurate. Nevertheless, since an early prediction is more useful than one that is made just a few seconds before, the value begins to fall rapidly. After the point B where the event actually happens, detecting an event is still useful and the value decreases slowly. The optimal prediction point A in Figure 3.1 motivates this thesis' search for a predictive event processing model.

Another difference between CEP and PA is the need for building rules that detect patterns. CEP relies heavily on predefined rules that have to be implemented by a domain expert that knows the complex event in question. This can be considered a weak point of CEP. PA, in turn, aims at automating the rule creation. Of course, PA methods need to be implement, too, but that happens before the model is crafted into a certain scenario. The model should then adapt to different kinds of scenarios. In CEP a specialized model is required for each different scenario.

There are several requirements that should be taken into consideration when designing a predictive CEP system. First, CEP and PA components must be able to communicate with each other so that CEP receives Primary Complex Event (PCE) predictions from PA and PA receives predictors from CEP. Second, integrating the PA component should not affect the existing CEP part nor its maintainability in any way. [21]

There are two ways to overcome these requirements. The first is to introduce predictive event processing agents (PEPAs) into the Event Processing Network (EPN). The second is to create a separate predictive event processing network (PEPN) that is somehow connected to the original EPN. Only the latter of these options satisfied the condition that the original EPN should not be affected in any way. The maintainability of EPN suffers greatly if PEPAs are mixed with original EPAs. [21] The implementation chapter describes how this requirement is actually fulfilled.

3.3 Predicting with Time Series Classification

Our task is to predict whether or not a certain event will happen in a defined time period in the future. Figure 3.2 shows an outline of this process. Let's assume that the complex event we are trying to predict occurs between $t_{E,1}$ and $t_{E,2}$. Then, in order to gain advantage from predictive analytics, the prediction should happen before the warning time begins at $t_{P,2}$. We can define a prediction time period to be the interval from $t_{P,1}$ to $t_{P,2}$, which contains the event history available for making the prediction. Thus, we can assume that the prediction happens at $t_{P,2}$ and uses information from $t_{P,1}$ to $t_{P,2}$. In Chapter 4 I will discuss how to choose the interval lengths

$$\begin{aligned}
\textbf{windowLength} &= t_{P,2} - t_{P,1} && (3.1)\\
\textbf{waitingInterval} &= t_{E,1} - t_{P,2} && (3.2)\\
\textbf{eventInterval} &= t_{E,2} - t_{E,1} && (3.3)
\end{aligned}$$

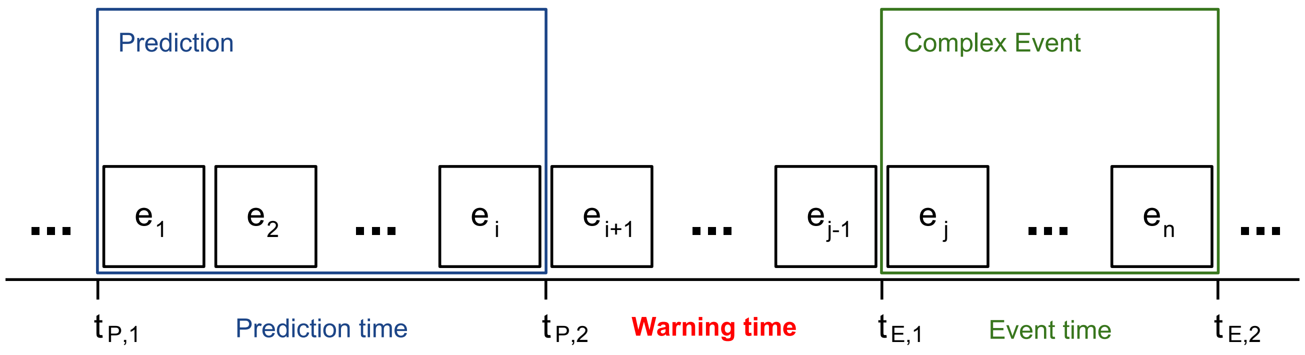

Figure 3.2: Timing of complex event prediction.

The event history $h = \{e_1, ..., e_i\}$ in the prediction interval is a multivariate time series, that is, it contains a sequence of numerical vectors. [64] Each vector contains the measurement values from different sensor at a certain point in time. I will design the system so that the CEP engine receives every Δt seconds a new event that contains the recent measurement values from all the available sensors. The choice of Δt depends on the phenomenon investigated, an issue which I will talk about more in Chapter 4.

We can formulate the prediction process as a binary classification task that contains two classes:

1. Histories that precede an event

2. Histories that do not precede an event

We can denote theses classes by w_1 and w_2. The task is to learn a classifier C, which maps a history h into a class w: $C \; : \; h \rightarrow w, \;\; w \in C$, where

$$\begin{aligned} C &= \{w_1, w_2\} \\ &= \{\text{“Event is not going to happen”}, \text{“Event is going to happen”}\} \end{aligned}$$

Time series classification methods can be divided into different categories: distance based methods, feature based methods, model based methods and so on. Two of these, distance based and feature based methods are investigated and tested in this thesis.

From distance based methods I present L_p norms and Dynamic Time Warping (DTW) as distance measures in co-operation with k-Nearest Neighbor (kNN) and Learning Vector Quantization as classification methods. The empirical section of this thesis employs a model that uses DTW and kNN.

From feature based methods I present Fourier and Wavelet analyses for feature extraction and then Support Vector Machines (SVMs) for classifying the feature vectors.

3.4 Distance Based Time Series Classification

Distance based methods are probably the simplest way to classify time series. They rely on a measure function that relates the time series to a single numeric value which indicates the similarity between two different time series. Then, a classification method can be used to separate different classes of time series. [64]

In this chapter I first present L_p norms and Dynamic Time Warping (DTW) as examples of measure functions. Then, k-nearest neighbors (kNN) and learning vector quantization (LVQ) algorithms are formulated for classifying the time series.

3.4.1 L_p norms

One way to classify time series is to calculate a distance measure between a new time series and the existing labeled time series. Then, one can select the group with has the lowest distance with the time series being classified. The easiest approach is some L_p norm which can be calculated for time series $\mathbf{x} = \{x_0, x_1, ..., x_{n-1}\}$ and $\mathbf{y} = \{y_0, y_1, ..., y_{n-1}\}$ with [53]

$$D(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=0}^{n-1} |x_i - y_i|^p \right)^{\frac{1}{p}}. \tag{3.4}$$

Then, one can use various algorithms, such as nearest neighbor classifiers or decision trees, to classify the time series into different groups. Examples of L_p norms are Manhattan norm ($p = 1$), Euclidean norm ($p = 2$) and maximum norm ($p \to \infty$). L_p norms are very straightforward to calculate but requires normalization of the signals in order to handle similar signals with different amplitudes. [46]

3.4.2 Dynamic Time Warping

L_p norms cannot group signals that are, for example, in different phases, no matter how similar they are. One way to overcome this problem is to use dynamic time warping (DTW) algorithm. It measures the similarity between two signals that may vary in time or speed by "warping" the axis of one time series so that the phases of the signals match. [46] Let us denote a feature space, that is, available values for x_i and y_j by $\mathcal{F}$. A classical DTW algorithm begins by creating a matrix

$$C(i, j) = c(x_i, y_j), \tag{3.5}$$

where c is a local distance measure: $c : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ that is the difference between the two variables. In other words, the two time series being compared are laid on the two axes of the matrix and the matrix contains the differences between the corresponding values in the time series.

Then, by using dynamic programming, a monotonically increasing minimal path from the bottom left corner of the matrix to the top right corner is searched. The algorithm begins by calculating the cumulative matrix D beginning from the bottom left corner of the cost matrix C. Then, the minimum path in the cumulative matrix defines the optimal alignment between x and y. [41]

The difference between Euclidean distance and DTW distance is illustrated in Figure 3.3. While Euclidean distance always compares the two time series point by point, DTW distance warps the other time series so that the extrema of the two time series are matched. This gives a more comprehensive similarity measure for two time series.

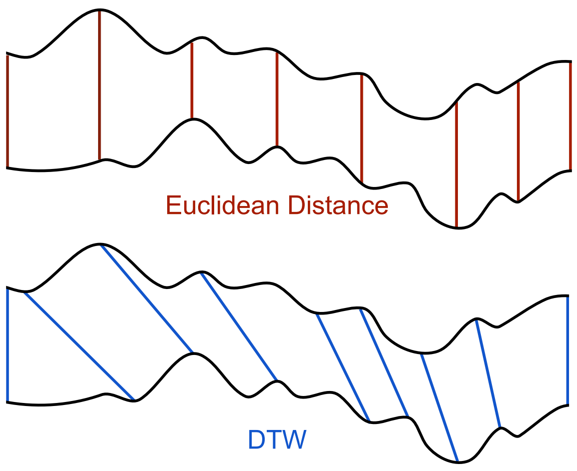

Figure 3.3: Difference between Euclidean Distance and Dynamic Time Warping.

The vertical lines in the figure illustrate the points in the time series that are compared against each other. As can be seen, the algorithm matches the peaks and the valleys with one another. A pseudo code is presented in the following:

```
Data: Cost function $c$, Width $N$, Height $M$
Result: Optimal Warping Path $p$
Calculate cumulative cost matrix D starting from bottom left corner;
Set n = N and m = M;
while $n > 0$ and $m > 0$ do
    Move to neighbor with the smallest D(n, m);
    Store movement to p;
    Update n and m;
end
reverse $p$;
return $p$;
```

3.4.3 k-Nearest Neighbor Algorithm

Our classifying task consists of two classes as shown in Equation 3.4. Once we have agreed on which distance measure we are using, we can use k-nearest neighbor (kNN) algorithm to classify new time-series instances. In the experimental chapter of this thesis I use Dynamic Time Warping (DTW) that was introduced in the previous section.

The kNN algorithm is a supervised learning algorithm, which means that we need a labeled teaching set to initialize the algorithm. [57] The time-series in the teaching set have been labeled so that their label is either w_1 or w_2 depending on which class they belong to.

Now for every new time-series instance we calculate the distance between the new instance and all the instances in the teaching set. Then, we select the k instances with the least distances to our new instance. These k instances vote for how the new instance is classified: the class with most instances is chosen. Clearly, the parameter k should be odd so that we avoid draws between the two classes. [28]

3.4.4 Learning Vector Quantization

Learning vector quantization (LVQ) is a supervised neural network that uses winner-take-all prototype-based learning. The LVQ algorithm, first developed by Teuvo Kohonen [34], is similar to the kNN except for the application of moving prototype vectors. The M prototype vectors $\{z_1, ..., z_M\}$ are labeled vectors that represent the classes $C(z_m), m = 1, 2, ..., M$ and can be selected randomly from the set of available training vectors. Then, for each training vector $x_i, i = 1, ..., N$ the nearest prototype vector z_m is updated with the following rule:

$$z_m \leftarrow \begin{cases} z_m + \alpha(x_i - z_m), & \text{if } z_m \text{ and } x_i \text{ belong to the same class} \\ z_m - \alpha(x_i - z_m), & \text{if } z_m \text{ and } x_i \text{ belong to different classes} \end{cases}, \tag{3.6}$$

where α is the learning rate. In other words, the closest prototype vector is moved towards the instance if its from the same class as the prototype vector. In the opposite case, the prototype vector is moved away from the instance. When using the model with testing data, each new instance is classified with the same class as the closest prototype vector.

Usually, the training vectors are iterated through multiple times for a better convergence. On every iteration the value of α can be decreased so the algorithm first takes larger steps and then gradually moves to smaller steps. By doing this it first moves the prototype vectors to correct areas and then refines their positions. [36]

3.5 Feature Based Time Series Classification

The aim of feature generation is to transform a sequence of time series data into a single vector that has as few dimensions as possible but still contains the relevant information about the time series for the classifying task. This process is a part of the preprocessing step of machine learning. We do this in order to reduce dimensionality, to increase learning accuracy and to improve result comprehensibility. [66]

There is a clear difference between feature selection and feature extraction. The former means choosing a subset from the available variables and using that as a feature vector, which is not an applicable method in the case of a time series as each variable in our multivariate time series is of a too high dimensionality by itself. The latter means mapping the available data into a lower dimensional space with some algorithm. [66] In sections 3.5.1 and 3.5.2 I present a technique for feature extraction by capturing a subset of the signal spectra with wavelet analysis.

Theoretically we could combine all the time series vectors from the prediction interval into one large feature vector that is then used as an input for the classifier C. However, as the prediction interval gets longer, we face the curse of dimensionality. The longer the feature vector is, the sparser the data becomes and the less statistically significant the classifying is. Also, the volume of the data increases rapidly and the processing becomes more and more demanding. [43] For this reason, I describe a way to reduce the dimensionality of the feature vectors.

3.5.1 Wavelet Analysis

Wavelet analysis is a generalization of Fourier analysis which breaks the signal into series of sines and cosines. This transformation reveals the frequency space properties but loses all the temporal information. [20] A Fourier transform of a function $x(t)$ is

$$F_x(t) = a_0 + \sum_{k=1}^{\infty} a_k cos(kt) + b_k sin(kt), \tag{3.7}$$

where

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} x(t)dt \tag{3.8}$$

$$a_k = \frac{1}{\pi}\int_0^{2\pi} x(t)cos(kt)dt \tag{3.9}$$

$$b_k = \frac{1}{\pi}\int_0^{2\pi} x(t)sin(kt)dt. \tag{3.10}$$

This corresponds to a mapping into the frequency space that is formed by orthogonal basis functions, sine and cosine. By orthogonality we mean that for a set of signals $\psi_n(t)$, $-\infty < n < \infty$, we have

$$\langle \psi_n(t), \psi_m(t) \rangle = 0, \; m \neq n \tag{3.11}$$

We can generalize the idea of Fourier transform into any orthogonal basis of signals $\psi_n(t)$. The analysis part calculates the coefficients of the original signal $x(t)$ in the new space:

$$c_n = \frac{\langle x(t), \psi_n(t) \rangle}{\langle \psi_n(t), \psi_n(t) \rangle}. \tag{3.12}$$

Then, the original signal can be constructed from the coefficients by

$$x(t) = \sum_{n=-\infty}^{\infty} c_n \psi_n(t), \tag{3.13}$$

which is called the synthesis process.

Fourier transform reveals the frequency band of the signal while losing all the temporal information about different frequencies. [20] Because of this Fourier transform is applicable only to stationary signals $f(t)$ whose variance does not vary with time. No information about the changes in variance are captured by Fourier transformation and hence the method becomes useless with non-stationary signals. Anomalies in time series data cause spectral variance and hence the signal is not stationary. [29]

We can define the short time Fourier transform (STFT) as

$$\text{STFT}(f,s) = \int_{-\infty}^{\infty} x(t)g(t-s)e^{-j2\pi ft}dt, \tag{3.14}$$

where f is frequency of $x(t)$ and $g(t)$ is a sliding window function, for example, a box function

$$\text{box}(t) = \begin{cases} 1 & \text{for } |t| \leq 1/2 \\ 0 & \text{elsewhere} \end{cases}. \tag{3.15}$$

This restricts the Fourier transform into one window at the time and thus achieves time-localization which the ordinary Fourier transform lacks of. For computational purposes we must discretize this by denoting $f_n = n/T$ and $s_m = mT$. In this case our orthogonal basis functions are

$$v_{n,m}(t) = e^{j2\pi nt/T}g(t-mT). \tag{3.16}$$

Now the analysis part is

$$c_{n,m} = \frac{\langle x(t), v_{n,m}(t)\rangle}{\langle v_{n,m}(t), v_{n,m}(t)\rangle} \tag{3.17}$$

$$= \frac{1}{T}\int_{mT}^{(m+1)T} x(t)e^{-j2\pi nt/T}dt \tag{3.18}$$

and the synthesis is

$$x(t) = \sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty} c_{n,m}e^{j2\pi nt/T}g(t-mT). \tag{3.19}$$

Unlike Fourier transform, wavelet transformation gives the signal localization in both time and frequency spaces. It does this by using a concept called multi-resolution analysis (MRA) which means that different frequencies are analyzed with different resolutions. While Fourier transformation uses equally-sized windows for all frequencies, Wavelet transformation uses longer time windows for low frequencies and shorter time windows for high frequencies. This approach allows good localization of high frequency components while preserving information about low frequency contents of the signal. [20]

Continuous wavelet transform (CWT) for signal $x(t)$ is defined as

$$C_x(a,b) = \frac{1}{\sqrt{|a|}}\int_{-\infty}^{\infty} x(t)w\left(\frac{t-b}{a}\right)dt, \tag{3.20}$$

where a and b are scaling and positioning parameters and $w(t)$ is a so-called mother wavelet function. [20]

For computational purposes we must find a discrete wavelet transform (DWT). By choosing the scales to be powers of 2 and the positions to be multiples of the scales, we get an orthogonal basis of functions for CWT:

$$w_{j,k}(t) = 2^{j/2} w(2^j t - k). \tag{3.21}$$

These $w_{j,k}(t)$ are called the baby wavelets. [44] As can be easily seen, baby wavelets become narrower and higher as the j increases. The reverse is also true, baby wavelets become wider and flatter as the j decreases. We can now define space W_j to contain all the signals $x_j(t)$ that can be synthesized from baby wavelets $w_{j,k}(t)$ with the j fixed:

$$x_j(t) = \sum_{k=-\infty}^{\infty} c_{j,k} w_{j,k}(t). \tag{3.22}$$

Similarly, let V_j to be the space of signals $x(t)$ that can be synthesized from baby wavelets $w_{i,k}(t)$ where $i < j$ and $-\infty < k < \infty$. Then, from the definition of W_j and V_j we have $V_{j+1} = W_j + V_j$. This is illustrated in figure 3.4.

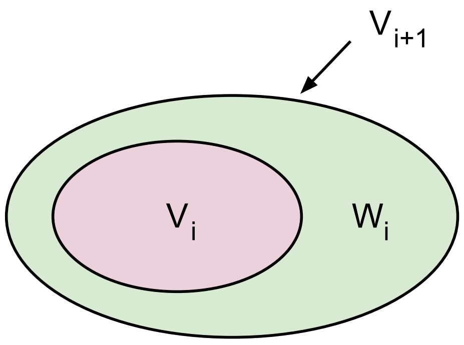

Figure 3.4: Wavelet subspaces.

Thus, the spaces V_j are nested inside each other, that is $\{0\} \subset \ldots \subset V_{-1} \subset V_0 \subset V_1 \subset \ldots L^2$, where L^2 contains all possible signals. The spaces W_j are the differences between adjacent spaces V_j and V_{j+1}. Now we can divide the space V_0 as

$$\begin{aligned} V_0 &= V_{-1} + W_{-1} \\ &= V_{-2} + W_{-2} + W_{-1} \\ &= V_{-3} + W_{-3} + W_{-2} + W_{-1} \\ &= \cdots \end{aligned}$$

and the signal as

$$\begin{aligned} x(t) &= A_1(t) + D_1(t) && (3.23) \\ &= A_2(t) + D_2(t) + D_1(t) && (3.24) \\ &= A_3(t) + D_3(t) + D_2(t) + D_1(t) && (3.25) \\ &= \cdots , && (3.26) \end{aligned}$$

where $D_i(t) \in W_{-i}$ is the detail at level i and $A_i(t) \in V_{-i}$ is the approximation at level i. This approach is called multi-resolution analysis (MRA) because on each step we divide the frequency band into two pieces and then continue the process on the lower half. At each stage the $A_i(t)$ corresponds to low-pass filtered signal and the $D_i(t)$ to high-pass filtered signal. This way we get more detailed information about the high frequencies. [44]

To facilitate the computations we can define a scaling function $\phi(t)$ (sometimes called the father wavelet) which produces the subspaces V_j:

$$\phi_{j,k}(t) = \sqrt{2^j}\phi(2^j t - k). \qquad (3.27)$$

Because $V_0 \subset V_1$ and $W_0 \subset V_1$, it is possible to construct the mother wavelet and the scaling function in V_1 from the scaling function in V_0 as follows

$$\begin{aligned} \phi(t) &= \sum_n h_0(n)\sqrt{2}\phi(2t - n) && (3.28) \\ w(t) &= \sum_n h_1(n)\sqrt{2}\phi(2t - n), && (3.29) \end{aligned}$$

where $h_0(n)$ and $h_1(n)$ are discrete time filter coefficients. These depend on the choice of wavelet type and they will be defined later. Now we are able to define the signal $x(t) \in V_j$ using the mother wavelet and the scaling function in spaces V_{j-1} and W_{j-1}

$$\begin{aligned} x(t) &= \sum_k cA_0(k)\phi_{j,k}(t) && (3.30) \\ &= \sum_k cA_1(k)\phi_{j-1,k}(t) + \sum_k cD_1(k)w_{j-1,k}(t) && (3.31) \\ &= A_1(t) + D_1(t), && (3.32) \end{aligned}$$

where $cA_0(k)$, $cA_1(k)$ and $cD_1(k)$ are the approximate and detail coefficients respectively. Then, $\phi_{j-1,k}$ can be further filtered to get the next level signals as shown in equation 3.23. The coefficient values can be derived as follows

$$\begin{aligned} cA_1(k) &= \langle x(t), \phi_{j-1,k}(t)\rangle && (3.33) \\ &= \left\langle \sum_n cA_0(n)\phi_{j,n}(t), \phi_{j-1,k}(t) \right\rangle && (3.34) \\ &= \sum_n cA_0(n) \left\langle \phi_{j,n}(t), \phi_{j-1,k}(t)\right\rangle , && (3.35) \end{aligned}$$

which, by calculating the inner product simplifies to

$$cA_1(k) = \sum_n h_0(n-2k)cA_0(n) \tag{3.36}$$

Similarly, for $cD_1(k)$ we get

$$cD_1(k) = \sum_n h_1(n-2k)cA_0(n) \tag{3.37}$$

These two operations correspond to filters

$$cA_0(n) \longrightarrow \boxed{h_0(-n)} \longrightarrow \boxed{\downarrow 2} \longrightarrow cA_1(k) \tag{3.38}$$

$$cA_0(n) \longrightarrow \boxed{h_1(-n)} \longrightarrow \boxed{\downarrow 2} \longrightarrow cD_1(k), \tag{3.39}$$

where the downsampling filter, $\boxed{\downarrow 2}$, means omitting every other value from the signal. As shown in equation 3.23, we can further use the downsampling filters to decompose the signal into more detail. This process is shown in Figure 3.5.

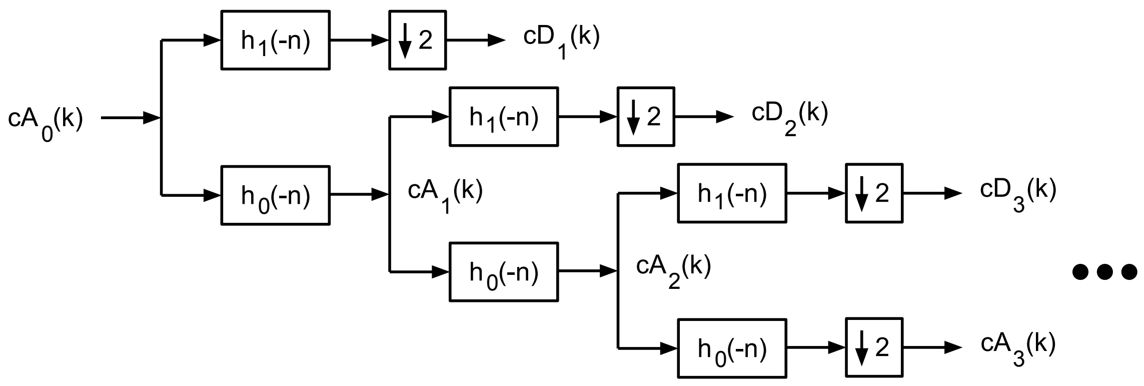

Figure 3.5: Wavelet coefficient decomposition using filters.

Now on level N our coefficients are

$$C = [cA_N, cD_N, cD_{N-1}, ..., cD_2, cD_1] \tag{3.40}$$

The signal is now compressed by setting some of the detail coefficients to zero. Of course, the energy of the signal is not completely preserved and the number of zero coefficients depends on the application.

Next, we need to perform the synthesis part by opposite, upsampling filters. This process is shown in Figure 3.6.

If no coefficients are set to zero in the analysis phase, the signal that is constructed in the synthesis phase is exactly the same as the original signal. By setting some of the detail coefficients to zero before the synthesis, some energy of the signal is lost but at the same time the signal is compressed to a smaller size.

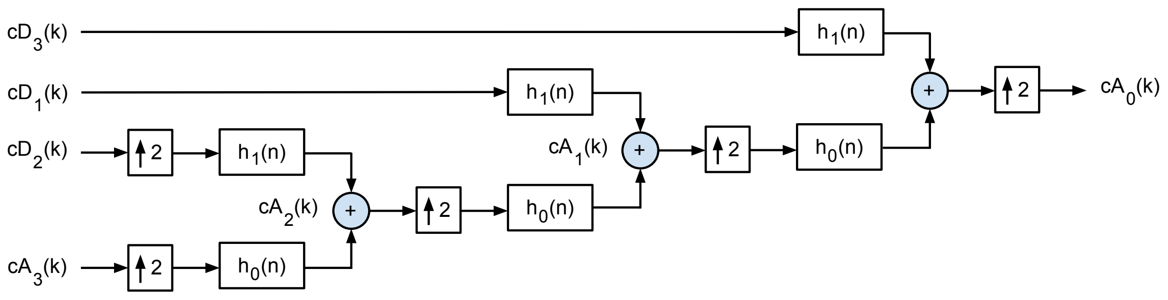

Figure 3.6: Wavelet coefficient decomposition using filters.

3.5.2 Haar Wavelet Decomposition for Time Series

Haar wavelet [55] is the simplest possible wavelet and it has the following mother wavelet

$$\psi(t) = \begin{cases} 1, & 0 \leq t < 1/2 \\ -1, & 1/2 \leq t < 1 \\ 0, & otherwise \end{cases} . \tag{3.41}$$

and the following scaling function

$$\phi(t) = \begin{cases} 1, & 0 \leq t < 1 \\ 0, & otherwise \end{cases} . \tag{3.42}$$

For Haar wavelets the discrete time filter coefficients $h_0(n)$ and $h_1(n)$ are defined as

$$h_0 = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right] \tag{3.43}$$

$$h_1 = \left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right] . \tag{3.44}$$

Now let's assume we have a time-series $\mathbf{x} = (x_1, x_2, ..., x_N)$ of length $N = 2^n$. The 1-level Haar-Transform is

$$\mathbf{x} \longleftrightarrow (A_1|D_1), \tag{3.45}$$

where

$$A_1 = \left(\frac{x_1 + x_2}{\sqrt{2}}, \frac{x_3 + x_4}{\sqrt{2}}, ..., \frac{x_{N-1} + x_N}{\sqrt{2}}\right) \tag{3.46}$$

$$D_1 = \left(\frac{x_1 - x_2}{\sqrt{2}}, \frac{x_3 - x_4}{\sqrt{2}}, ..., \frac{x_{N-1} - x_N}{\sqrt{2}}\right) . \tag{3.47}$$

The former operation corresponds to a running average (trend) and the latter to a running difference (fluctuation). Then, A_1 can be further decomposed into A_2 and D_2 by performing the same operation again:

$$\mathbf{x} \longleftrightarrow (A_2|D_2|D_1). \tag{3.48}$$

This process is then repeated until a desired level is reached. The Wavelet can be compressed by setting some of the detail coefficients to zero. The resulting vector can now be used as a feature. In the next chapters I describe how these features can be classified with first linear classifiers and then with support vector machines (SVMs) that generalize into nonlinear cases.

3.5.3 Linear Classifiers

In this section I discuss the special case of linearly separable classes. If the input space is linearly separable, we can use a linear hyperplane to separate the two classes. [56] As explained in Chapter 3.3, we have two classes, w_1 and w_2, for which the l-dimensional hyperplane is

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0, \tag{3.49}$$

where $\mathbf{x}$ is a feature vector, $\mathbf{w}$ is the weight vector, and w_0 is the threshold. The threshold value is needed to cover the case of the hyperplane not crossing the origin. We can extend the vectors into $(l+1)$-dimensional space by defining

$$\mathbf{x}' = \left[\mathbf{x}^T, 1\right]^T \tag{3.50}$$

$$\mathbf{w}' = \left[\mathbf{w}^T, w_0\right]^T. \tag{3.51}$$

From now on the variables $\mathbf{x}$ and $\mathbf{w}$ refer to $\mathbf{x}'$ and $\mathbf{w}'$. Now the feature vectors x have the properties

$$\mathbf{w}^T\mathbf{x} > 0 \qquad \forall \mathbf{x} \in w_1 \tag{3.52}$$

$$\mathbf{w}^T\mathbf{x} < 0 \qquad \forall \mathbf{x} \in w_2. \tag{3.53}$$

The problem is now the choice of the weight vector w. One way to tackle the problem is the perception algorithm which defines a perception cost as

$$J(\mathbf{w}) = \sum_{\mathbf{x} \in Y} (\delta_x \mathbf{w}^T\mathbf{x}), \tag{3.54}$$

where Y is the set of training vectors that are misclassified with the weigh vector w and $\delta_x = -1$ if $\mathbf{x} \in w_1$ and $\delta_x = +1$ if $\mathbf{x} \in w_2$. Since the cost $J(\mathbf{w})$ is always positive, continuous and piecewise linear, it can be minimized with a gradient descent method [56]:

$$\mathbf{w(t+1)} = \mathbf{w(t)} - \rho_t \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \tag{3.55}$$

$$= \mathbf{w(t)} - \rho_t \sum_{\mathbf{x} \in Y} \delta_x \mathbf{x}. \tag{3.56}$$

The iteration is performed until all the training instances have been classified correctly. A variation of perception algorithm loops through the training instances one by one and updates the weight vector on each instance.

Even though the classes are not linearly separable, we can find an optimal linear classifier that minimizes the classification error in some sense. One way to find the optimal w is to use mean square error estimation which tries to minimize the cost function

$$J(\mathbf{w}) = E[|y - \mathbf{x}^T\mathbf{w}|^2], \tag{3.57}$$

where y is the desired output; in this case $y = 1$ for w_1 and $y = -1$ for w_2. [23] A required condition for the minimum is

$$\frac{\partial J(\mathbf{w}}{\partial \mathbf{w}} = 2E[\mathbf{x}(y - \mathbf{x}^T\mathbf{w})] = 0. \tag{3.58}$$

The optimal weigh vector is then

$$\mathbf{w} = R_x^{-1}E[\mathbf{x}y], \tag{3.59}$$

where R_x is the correlation matrix of $\mathbf{x}$ and $E[\mathbf{x}y]$ is the cross-correlation vector that have to be estimated from the learning set.

Another way is to use least squares methods that use the cost function

$$J(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{x}_i^T\mathbf{w})^2 = \sum_{i=1}^{N} e_i^2 \tag{3.60}$$

Then, by differentiating with respect to $\mathbf{w}$ we get

$$\sum_{i=1}^{N}\mathbf{x}_i(y_i - \mathbf{x}_i^T\mathbf{w}) = 0 \tag{3.61}$$

$$\Leftrightarrow \quad (\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T)\mathbf{w} = \sum_{i=1}^{N}(\mathbf{x}_i y_i) \tag{3.62}$$

$$\Leftrightarrow \quad (\mathbf{X}^T\mathbf{X})\mathbf{w} = \mathbf{X}^T\mathbf{y} \tag{3.63}$$

$$\Leftrightarrow \quad \mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \tag{3.64}$$

where $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$, $\mathbf{y} = [y_1, ..., y_N]^T$, and $\mathbf{X}^T\mathbf{X}$ is sample correlation matrix which can be estimated from the available data. [23]

3.5.4 Support Vector Machines for Linearly Separable Classes

The following three chapters follow the support vector machine chapters 3.7 and 4.18 discussed in the book *Pattern recognition* by Theodoridis and Koutroumbas [57].

I first discuss the Support Vector Machines in case of two linearly separable classes. Similarly to the previous section with linear classifiers, we design a hyperplane

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0 \tag{3.65}$$

When the hyperplane leaves maximal margins between the class instances and itself, the performance of classifier is optimal. This situation is depicted in Figure 3.7.

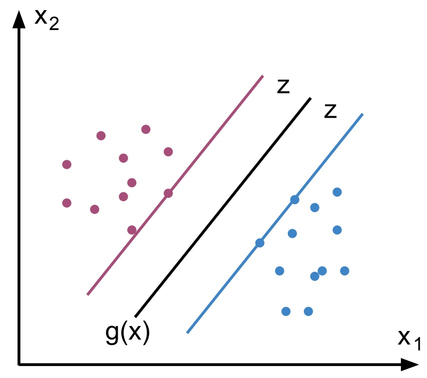

Figure 3.7: Maximum separating hyperplane between two linearly separable classes.

Thus, we should maximize the margin z which can be represented as $z = |g(\mathbf{x})|/||\mathbf{w}||$. If we scale $\mathbf{w}$ and w_0 so that $g(\mathbf{x}) = 1$ for the class w_1 and $g(\mathbf{x}) = -1$ for the class w_2 on the points that are closest to the hyperplane, we have a margin of $2/||\mathbf{w}||$ and requirements

$$\mathbf{w}^T\mathbf{x} + w_0 \geq 1, \forall\mathbf{x} \in w_1 \tag{3.66}$$

$$\mathbf{w}^T\mathbf{x} + w_0 \leq -1, \forall\mathbf{x} \in w_2 \tag{3.67}$$

We can form the problem as the following optimization problem

$$\text{minimize } J(\mathbf{w}, w_0) = \frac{1}{2}||\mathbf{w}||^2 \tag{3.68}$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1,\ i = 1, 2, ..., N \tag{3.69}$$

where $y_1 = 1$ and $y_2 = -1$. The Karush-Kuhn-Tucker (KKT) conditions for this nonlinear quadratic optimization problem are

$$L_{\mathbf{w}}(\mathbf{w}, w_0, \lambda) = \mathbf{0} \tag{3.70}$$

$$L_{w_0}(\mathbf{w}, w_0, \lambda) = 0 \tag{3.71}$$

$$\lambda_i \geq 0,\ i = 1, 2, ..., N \tag{3.72}$$

$$\lambda_i[y_i(\mathbf{w}^T\mathbf{x}_i + w_0) - 1] = 0, i = 1, 2, ..., N, \tag{3.73}$$

where $L(\mathbf{w}, w_0, \lambda)$ is the Lagrangian function

$$L(\mathbf{w}, w_0, \lambda) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\lambda_i[y_i(\mathbf{w}^T\mathbf{x}_i + w_0) - 1] \tag{3.74}$$

Solving the first order differential equations in the KKT conditions yields a dual problem

$$\begin{aligned}
&\text{maximize} && L(\mathbf{w}, w_0, \lambda) && (3.75)\\
&\text{subject to} && \mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i && (3.76)\\
& && \sum_{i=1}^{N} \lambda_i y_i = 0 && (3.77)\\
& && \lambda \geq \mathbf{0} && (3.78)
\end{aligned}$$

By substituting the first two conditions into the Lagrangian L we further get

$$\begin{aligned}
&\max_{\lambda} && \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right) && (3.79)\\
&\text{subject to} && \sum_{i=1}^{N} \lambda_i y_i = 0 && (3.80)\\
& && \lambda \geq 0 && (3.81)
\end{aligned}$$

As we can easily see, the function to be maximized does not depend on the dimensionality of the input space because it contains the input vectors in the form of a inner product. This allows us to easily generalize the method into non-separable classes by mapping them into a higher-dimensional space in the next sections.

The Support Vectors are the points $\mathbf{x}_i$ for which the Lagrangian multiplier λ_i is not zero and which lie on one of the hyperplanes $\mathbf{w}^T\mathbf{x} + w_0 = \pm 1$. The optimal hyperplane of SVM is unique because the cost function in equation 3.70 is strictly convex.

3.5.5 Support Vector Machines for Linearly Non-separable Classes

Now we turn to the case of classes that cannot be separated with a linear hyperplane. In this case we introduce slack variables $\xi_i \geq 0$ that allow some points to fall on the wrong side of the hyperplane. Now the equation 3.69 is replaced with

$$y_i[\mathbf{w}^T\mathbf{x} + w_0] \geq 1 - \xi_i, \tag{3.82}$$

where the value of ξ_i defines the type of the point as follows:

1. $\xi_i = 0 \Rightarrow$ Point is classified correctly.

2. $0 < \xi_i \leq 1 \Rightarrow$ Point is classified correctly but it is on the wrong side of the margin.

3. $\xi_i > 1 \Rightarrow$ Point is misclassified.

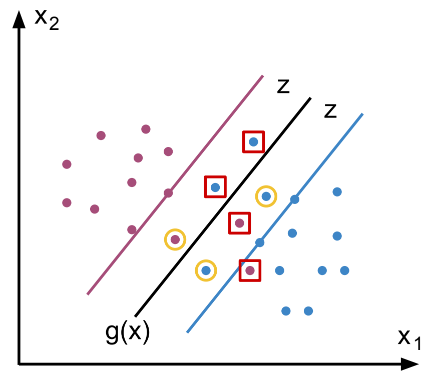

Figure 3.8: A separating hyperplane between two classes that are non-separable. A yellow ring marks a point that is classified correctly but that is on the wrong side of the margin. A red square marks a point that is misclassified.

These three types are illustrated in figure 3.8. An yellow ring marks a point that is classified correctly but that is on the wrong side of the margin (type 2). A red square marks a point that is misclassified (type 3).

In this case our optimization problem can be stated as

$$\text{minimize } J(\mathbf{w}, w_0, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N} I(\xi_i) \tag{3.83}$$

$$\text{subject to } y_i[\mathbf{w}^T\mathbf{x}_i + w_0] \geq 1 - \xi_i, i = 1, 2, ..., N \tag{3.84}$$

$$\xi_i \geq 0, i = 1, 2, ..., N, \tag{3.85}$$

where $I(\xi_i) = 1$ if $\xi_i > 0$ and $I(\xi_i) = 0$ if $\xi_i = 0$ and the parameter C defines a trade-off between maximizing the margin and minimizing the number of misclassified points. Similarly to the previous section, we can build a dual problem that leads to the problem

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right) \tag{3.86}$$

$$\text{subject to } 0 \leq \lambda_i \leq C, i = 1, 2, ..., N \tag{3.87}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0. \tag{3.88}$$

Again, the dimensionality of the input space disappears from the problem and the input vectors appear only as inner products. There is a shortcut for calculating these inner products with a kernel trick [31], which is described in the next section.

3.5.6 Generalization of SVMs into Nonlinear Cases

If the classes are not separable in the l-dimensional input space, we can find a mapping $\mathbf{x} \in \mathbb{R}^l \rightarrow \mathbf{y} \in \mathbb{R}^k$, where $k > l$. The choice of k is done so that the

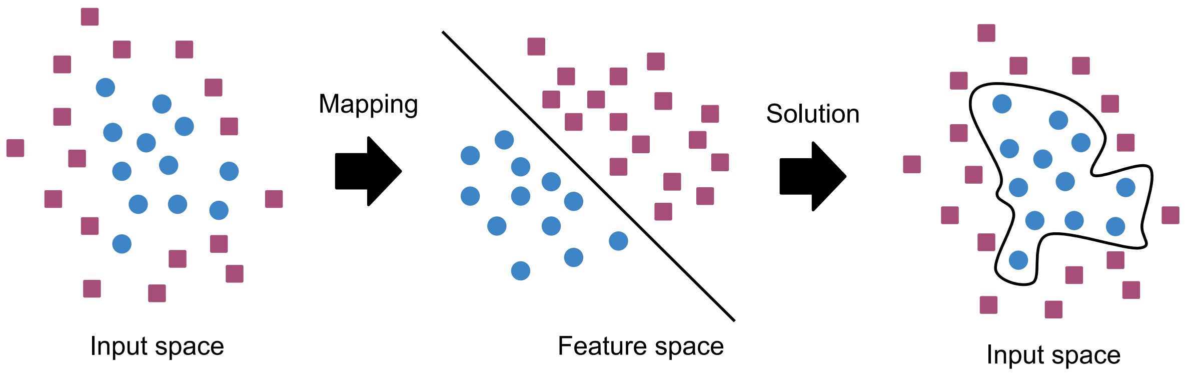

Figure 3.9: The main idea of Support Vector Machines. Classes are not linearly separable in the input space. They are mapped into a higher-dimensional space where a linear classifier can be formed. This classifier corresponds to a non-linear classifier in the original input space.

original classes are linearly separable in the new, higher-dimensional space. This situation is depicted in Figure 3.9. As mentioned in the previous chapters, our optimization problems contain input vector x only as an inner product that is not altered by the mapping.

In order to present the Mercer's Theorem [31] that relates the inner product to a kernel function, we need to explain some space related concepts. A Cauchy sequence is defined as a sequence $a_1, a_2, ...$ for which

$$\forall \epsilon \in \mathbb{R} \; \exists \; N > 0 \; s.t. \; |a_m - a_n| < \epsilon, \; m, n > N \tag{3.89}$$

In other words, the elements of the sequence become arbitrarily close as the sequence progresses. A complete space is a space where every Cauchy sequence converges to a point that is also contained within the space. A Hilbert space, in turn, is a generalization of Euclidean space into any finite or infinite number of dimensions. It is a complete vector space that has inner product defined for measuring lengths and angles. [12]

Mercer's Theorem states that for each mapping $\mathbf{x} \to \phi(\mathbf{x}) \in H$, where H is a Hilbert Space, we have a kernel function $K(\mathbf{x}, \mathbf{y})$ that is equivalent for the inner product:

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = K(\mathbf{x}, \mathbf{y}). \tag{3.90}$$

The kernel function must have the following properties

$$\int_S \int_S K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0 \tag{3.91}$$

$$\int_S g(\mathbf{x})^2 d\mathbf{x} < +\infty, \; \forall g(\mathbf{x}), \mathbf{x} \in S, \tag{3.92}$$

where $S \subset \mathbb{R}^l$. The opposite is also true: for any Kernel function satisfying the conditions above there is a Hilbert space where the Kernel function is equivalent for the inner product. The problem is now how to find the mapping ϕ when we have selected the Kernel function.

Now we can replace the inner product in Equation 3.79 with the Kernel function to get the optimization problem

$$\max_{\lambda} \left(\sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right) \tag{3.93}$$

$$\text{subject to } 0 \leq \lambda_i \leq C, \ i = 1, 2, ..., N \tag{3.94}$$

$$\sum_i \lambda_i y_i = 0. \tag{3.95}$$

The resulting non-linear classifier is then

$$g(\mathbf{x}) = \sum_{i=1}^{N_s} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + w_0 \begin{cases} > 0, \ \Rightarrow \mathbf{x} \in w_1 \\ < 0, \ \Rightarrow \mathbf{x} \in w_2, \end{cases} , \tag{3.96}$$

which is shown in Figure 3.10. Input vectors enter the network from the left. Then, inner products are calculated in middle nodes, the number of which is N, the number of support vectors. An output node sums the components up and outputs a single real number that determines the result of the classification according to equation 3.96.

3.5.7 Using SVM for Time Series Analysis

Before we can use our SVM classifier, we have to choose an appropriate kernel function $K(\mathbf{x}, \mathbf{y})$ for classifying wavelet based feature vectors. In literature one of the most used kernels is the Gaussian kernel [67], which is of the form

$$K(\mathbf{x}, \mathbf{y}) = \exp \frac{||\mathbf{x} - \mathbf{y}||^2}{2\gamma^2}, , \tag{3.97}$$

where $\beta > 0$ is a parameter chosen by the user. Gaussian kernel belongs to a family of functions called radial basis functions (RBF) whose value depends only on the distance between two points [50]

$$f(\mathbf{x}, \mathbf{y}) = f(||\mathbf{x} - \mathbf{y}||) \tag{3.98}$$

If we assume that the time series is generated by an AR(1)-process

$$x_T = g(x_{T-1}, ..., x_{T-k}) + \mu, \tag{3.99}$$

it can be shown that an ellipsoid with mean equal to the mean of the time series and variance equal to μ contains most of the data. The variable μ is a Gaussian noise component. Consequently, similar time windows are close to each other in the sense of Euclidean Distance. This makes RBF Kernels promising for time series analysis. [47]

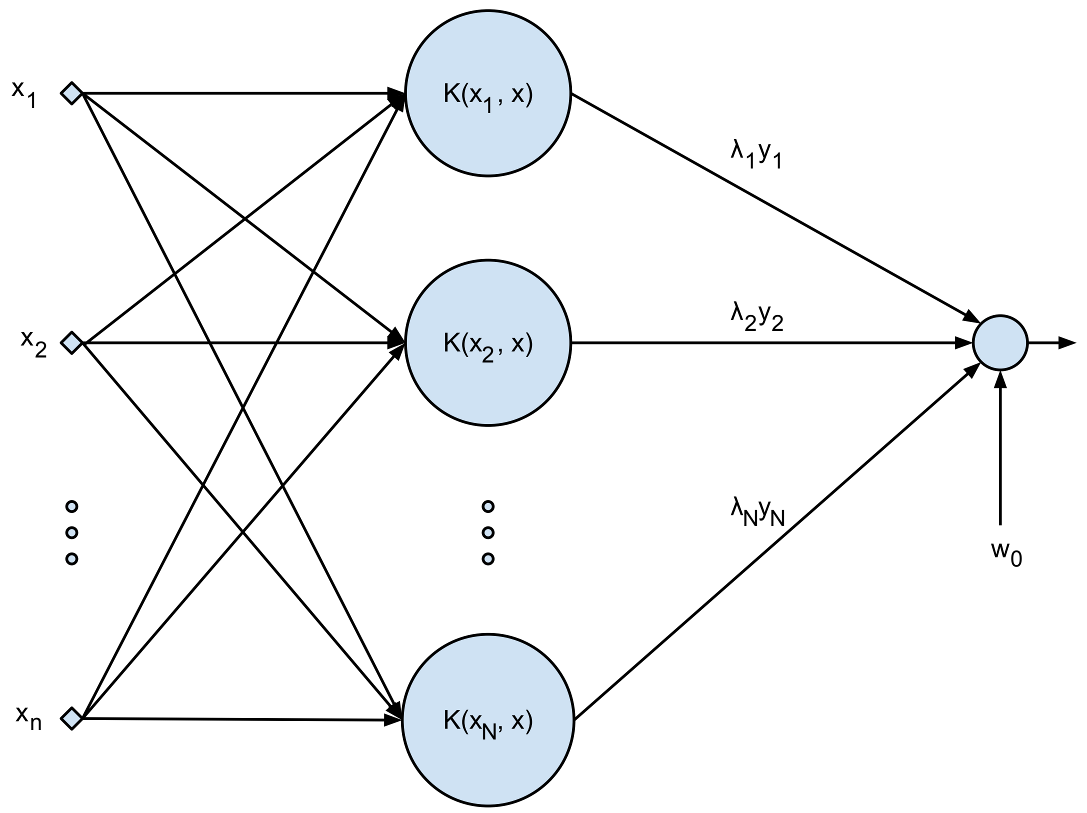

Figure 3.10: SVM classifier as a network. Input vectors enter the network from the left. Kernel functions calculate inner products in the middle nodes. Output node combines sums the components up and outputs a single real number.

Another possibility is Polynomial Kernel that is of the form

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^d, \tag{3.100}$$

which gives the Linear Kernel as a special case when $d = 1$. [50]

Subsequence Kernels look for informative subsequences that are similar in dependent time windows. In this case the Kernel function becomes

$$K(\mathbf{x}, \mathbf{y}) = \sum_{s_x, s_y} K(s_x, s_y), \tag{3.101}$$

where s_x and s_y are subsequences of x and y. There are $\binom{n}{k}^2$ possible combinations for time series of length n and subsequence of length k, so one must use another algorithm for selecting the optimal subsequences. [47]

In thesis I use Gaussian RBF Kernels because of their widespread use and good applicability to time series analysis. Ruping [47] compared Linear, RBF, Fourier, Subsequence and Hidden Markov Model Kernels and came into conclusion that RBF Kernels perform very well on time series learning tasks. However, he recommends to investigate other possibilities in very specialized applications.

A well defined kernel function increases the performance of the SVM greatly and thus Kernel selection is an important part of SVM development [20]. In the case of RBF Kernels we must choose the value of γ in 3.97. From now on I try to find an optimal value for σ, which then defines the γ from $\gamma = 1/2\sigma^2$. The value of σ should be chosen so that it minimizes the error

$$E(\sigma) = \sum_{i=1}^{N} |g(\mathbf{x}_i, \sigma) - y_i|^2 \tag{3.102}$$

A related theorem states that for a Support Vector Classifier (SVC) there exists a range $[\sigma_A, \sigma_B]$ for which

$$\forall\, \epsilon > 0 \ \text{ and } \ \forall\, \sigma_1, \sigma_2 \in [\sigma_A, \sigma_B] \tag{3.103}$$

$$\left| \sum_{i=1}^{N} |g(\mathbf{x}_i, \sigma_1) - y_i| - \sum_{i=1}^{N} |g(\mathbf{x}_i, \sigma_2) - y_i| \right| < \epsilon, \tag{3.104}$$

where $g(\mathbf{x}_i, \sigma_i)$ is the discriminant function from Equation 3.96 and y_i is the desired output (± 1) [63]. In other words, there is a range for σ where the Gaussian Kernel's generalization performance is stable.

To summarize our parameter estimation problem, we have two unknown parameters, Gaussian Kernel parameter γ and SVM weighting parameter C from 3.83. Now there are two ways to find the optimal values for the parameters: gradient search and grid search. Staelin [54] compared these two methods and came into the conclusion that they both achieve similar results in terms of accuracy. The only difference is clearly the computational cost that is much greater for the grid search.

To keep things simple, only the grid search is employed in this thesis. The grid is formed as follows

$$\ln\gamma \in \{\ln\gamma_0 - a_0, \ln\gamma_0 - a_0 + 1, ..., \ln\gamma_0, ..., \ln\gamma_0 + a_0\} \tag{3.105}$$

$$\ln C \in \{0, 1, ..., C_0\}, \tag{3.106}$$

where the parameters γ_0 and C_0 have to be chosen. Jaakkola's heuristics [30] give us a initial guess for γ_0. Let S be our training set. Then, from the set

$$G = \{\|\mathbf{x}_i - \mathbf{x}_j\| \mid (\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j) \in S, y_i \neq y_j\} \tag{3.107}$$

we can compute

$$\sigma_0 = \sigma_{Jaakkola} = \text{median}(G) \tag{3.108}$$

and finally $\gamma_0 = 1/\sigma_0^2$. The parameters a_0 and C_0 are chosen in the implementation chapter.

3.6 Parameter Selection and Model Validation

Our classifiers have several parameters that have to be chosen before using the models. Moreover, we need to somehow assess the quality of the classifier with the given parameter values so that an optimal model can be found.

In Chapter 3.6.1 I discuss how the available data should be divided into training and test data, in Chapter 3.6.2 I present some basic measures for the classifier's performance and in Chapter 3.6.3 I shortly review the requirements for system's computational performance.

3.6.1 Cross Validation

If the available training set is used for both estimating the parameters and validating the model, a problem called over-fitting will most probably be faced. It means that the model learns the training set "too well" and doesn't necessarily generalize and perform well when facing totally new instances. To tackle this problem we can use k-fold cross validation whose pseudo code goes as follows [15]:

```
Data: Training set $S$, integer $k$
Result: Performance measure
partition S into k disjoint equal-sized subsets $S_1, ..., S_k$;
performances = empty array of length $k$;
for $i = 1 \to k$ do
    $T = S \setminus S_i$;
    train model with teaching set $T$;
    performances[i] = model performance with test set $S_i$;
end
return average of values in performances array;
```

In other words, one part of the training data is separated at a time and used only for testing. Then, the final performance measure is the average of the different teaching and testing data sets.

3.6.2 Performance Measures

In this thesis our classifying task consists of two classes; w_1 contains time-series that do not precede a complex event and w_2 contains those that do. For both classes we either classify an instance correctly or not. Thus, we have $2 \times 2 = 4$ possibilities that are in a tabular form

		Predicted class		Total Instances
		+	-	
Actual class	+	TP	FN	P
	-	FP	TN	N

In the table above the positive (+) cells correspond to the class w_2 and the negative (-) cells to the class w_1. TP (True Positives) is the number of positive instances that are classified as positive. FP (False Positives) is the number of positives instances that are incorrectly classified as negative. Similarly, TN and FN are the numbers of correctly and incorrectly classified negatives. [45]

Clearly, we have identities $P = TP + FN$ and $N = FP + TN$. Now we can define further measures that are derived from the values in the previous table:

$$\textbf{True Positive Rate (TPR)} \text{ or } \textbf{Recall} = \mathrm{TP}/\mathrm{P} \tag{3.109}$$

$$\textbf{False Positive Rate (FPR)} = \mathrm{FP}/\mathrm{N} \tag{3.110}$$

$$\textbf{True Negative Rate (TNR)} = \mathrm{TN}/\mathrm{N} \tag{3.111}$$

$$\textbf{False Negative Rate (FNR)} = \mathrm{FN}/\mathrm{P} \tag{3.112}$$

$$\textbf{Precision} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FP}) \tag{3.113}$$

$$\textbf{Accuracy} = (\mathrm{TP}+\mathrm{TN})/(\mathrm{P}+\mathrm{N}) \tag{3.114}$$

$$\textbf{Error Rate} = (\mathrm{FP}+\mathrm{FN})/(\mathrm{P}+\mathrm{N}) \tag{3.115}$$

By varying model parameters we can construct a ROC (receiving operator characteristics) curve which is a sensitivity versus (1 - specificity) plot. Sensitivity is a synonym for TPR and (1 - specificity) corresponds to FPR, both of which are between 0 and 1. A completely random classifier achieves a ROC curve that is a straight line from (0,0) to (1,1). Classifiers with better performance are found above this line while worse performance results in a point below this line. [45] An example of a ROC curve is illustrated in Figure 3.11.

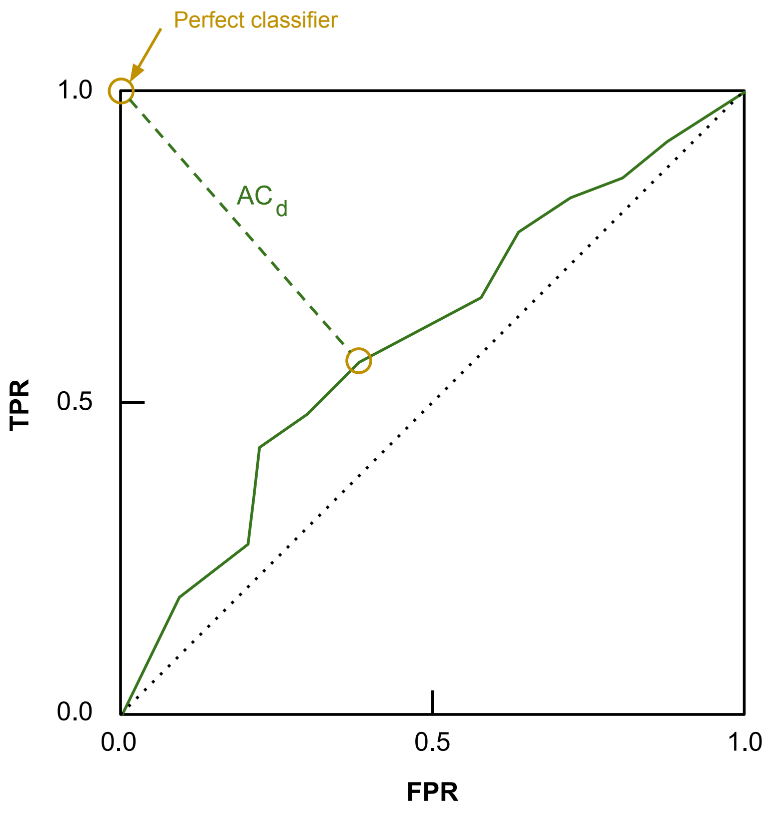

Figure 3.11: ROC (receiving operator characteristics) curve. The AC_d is the distance from a classifier's performance to that of a perfect classifier.

We still lack a single measure that could be used for ranking different classifiers. Since a perfect classifier is the point (0,1) on the ROC curve, we can rank the classifiers based on their distance to that point. Euclidean distance is defined as

$$\mathrm{AC}_d = \sqrt{W \cdot (1 - \mathrm{TPR})^2 + (1 - W) \cdot \mathrm{FPR}^2}, \tag{3.116}$$

where the parameter $W \in [0, 1]$ assigns relative importance to false positives and false negatives. The value of AC_d ranges from 0 for a perfect classifier to $\sqrt{2}$ for a completely incorrect one. [26]

To summarize, the parameter selection process goes like this:

1. Get parameters values one by one from the grid of available values
2. Calculate performance using cross validation and AC_d as performance measure
3. Choose the parameters values with the best performance (highest accuracy or lowest AC_d)

3.6.3 Computational Performance

Computational performance is not one of the key concerns of this thesis but it is still worth reviewing the possible bottlenecks and ways to improve performance. Not only can the amount of data be huge in CEP, but also the latency with which it is moving should be minimal. That said, our system is of no use if it cannot keep up with new sensor data coming in.

At the core of our system is the Esper CEP Engine which, according to its documentation [16], can manage over 500,000 events per second on a dual-core CPU 2 GHz processor with engine latency being less than 3 microseconds on average. The processor on our own test machine is a quad-core CPU 2.5 GHz processor so the performance should be at least at the same level.

It is important to note that training the model does not need to be real-time. In a real-life application the model is trained with regular intervals as a background process. While a new model is being trained, the old model can continue predicting. When the new model is ready, the old model is replaced with it. For example, in our test house this could be done once a month when the indoor air conditions have changed because of, for instance, seasonal changes in the weather.

The prediction phase, in turn, has to be fast as new measurements are coming in all the time. With our DTW and kNN model the prediction includes calculating the DTW distance with each of the training vectors and then selecting k closest ones. With our Wavelet and SVM model the prediction phase consists of the Haar Wavelet transform that produces a feature vector and support vector evaluation that classifies the feature vector.

Our computation performance tests evaluate the models' time complexity with respect to window sizes and the number of testing samples. Also the average execution times for classifying a single instance are measured.

Chapter 4

Implementation and Experiments

4.1 Case Study

4.1.1 ASTEKA Project

A member of the MMEA project, the research group of Environmental Informatics at the University of Eastern Finland (Kuopio campus) has a project called ASTEKA. The project aims at commercializing a monitoring system of living health and energy efficiency. [35] Basically, the project equips houses with sensors that provide informative online data presentations for the resident of the house. By monitoring different variables, such as energy consumption and CO_2 levels, the house or the resident could adjust the control variables accordingly.

Their first pilot was introduced at a living fare in Kuopio in the summer of 2010. The pilot originally consisted of 11 houses each of which has sensors ranging from CO_2 concentration and temperature to electricity and water consumption. The web interface of the pilot shows daily, weekly and monthly consumption statistics and charts which can be used to improve energy efficiency. [35] For example, one can easily notice whether the sudden peak in electricity consumption is caused by falling outside temperature or excessive oven usage.

4.1.2 Test House and Sensors

In this thesis I use data from one of the pilot houses. The house has three floors and sensors in all of them. The layout of the first floor is presented in Figure 4.1. The squares with numbers in them are sensor units which are listed in Table 4.1.

The second floor, which has only 2 sensor units, is shown in Figure 4.2 and its sensors are listed in Table 4.2. Both the two sensors in the second floor are located in bedrooms. However, bedroom 3 serves as a guest room and thus is empty most of the time, which should be detected in the data easily.

In the basement most of the sensors are located in the boiler room which is a well isolated space with no windows. The layout of the basement is presented in Figure 4.3. The listing of the sensors in the basement is shown in Table 4.3. An interesting detail in the basement is how the sensor unit 10 with humidity and

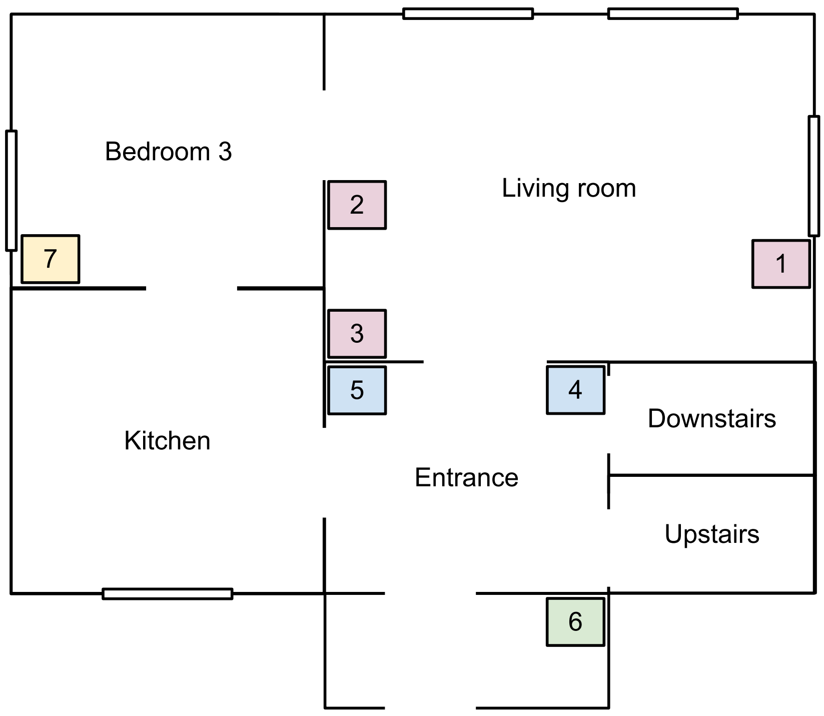

Figure 4.1: Layout of the first floor and the locations of the sensor units.

temperature sensors is placed near the showers and the sauna. This should definitely be shown somehow in the data from these sensors.

4.1.3 Test Data

The data from ASTEKA server is available via a Java library. The library allows one to specify two dates and it downloads two types of files, sensor files and measurement files. The sensor files are of the following form:

Filename: sensors8_11.txt

Header: SensorID:int; name:varchar(100)

Example line: 51;Temperature Fileplace (Har 1)

The file name contains to identification numbers: the former is a house ID and the latter is sensor group ID. The header line specifies the columns used in the file but since they are similar in each sensor file, they do not need to be read. The example line shows that the columns are separated by a semicolon.

The measurement files are of the following form:

Filename: measurements8_11.txt

Header: unixtime(BIGINT); ID_8_11_51(DOUBLE), ID_8_11_52(DOUBLE); ...

Table 4.1: Sensors in the first floor of the test house

First floor

Room	**Group**	**Sensor ID**	**Variable**	**Unit**
Living room	1	51	Temperature	$^{\circ}C$
Living room	2	52	CO	ppm
Living room	3	815	PIR	ON/OFF
		815	Humidity	%RH
		817	Temperature	$^{\circ}C$
		818	CO_2 OH	ppm
Inner entrance	4	53	Temperature ET	$^{\circ}C$
		54	Humidity ET	%RH
		55	CO_2	ppm
		60	Electricity	Wh
Inner entrance	5	844	Small particles	count / ft^3
		845	Large particles	count / ft^3
Outer entrance	6	56	Diff Pressure	Pa
Bedroom 3	7	650	Temperature	$^{\circ}C$
		651	Humidity	%RH
		652	CO_2	ppm
		789	VOC	ppm

Table 4.2: Sensors in the second floor of the test house

Second floor

Room	**Group**	**Sensor ID**	**Variable**	**Unit**
Bedroom 1	8	644	Humidity	%RH
		645	Temperature	$^{\circ}C$
		646	CO_2	ppm
Bedroom 2	9	647	Humidity	%RH
		648	Temperature	$^{\circ}C$
		649	CO_2	ppm

Table 4.3: Sensors in the basement of the test house

Basement

Room	**Group**	**Sensor ID**	**Variable**	**Unit**
Dressing room	10	57	Humidity	%RH
		58	Temperature	$°C$
		59	CO_2	ppm
Outside	11	772	Temperature	$°C$
Boiler room	12	62	Water	m^3
		773	Floor heating outgoing water	$°C$
		775	Floor heating	$°C$
		776	L1 indoor temperature, incoming	$°C$
		777	Used water	$°C$
		778	Variable	$°C$
		779	Radiator outgoing water	$°C$
		783	Hot water	%
		784	Radiator indoor temperature	$°C$
		785	District heating incoming temperature	$°C$
		786	District heating return temperature	$°C$
		787	District heating energy	kWh
		788	District heating water flux	m^3

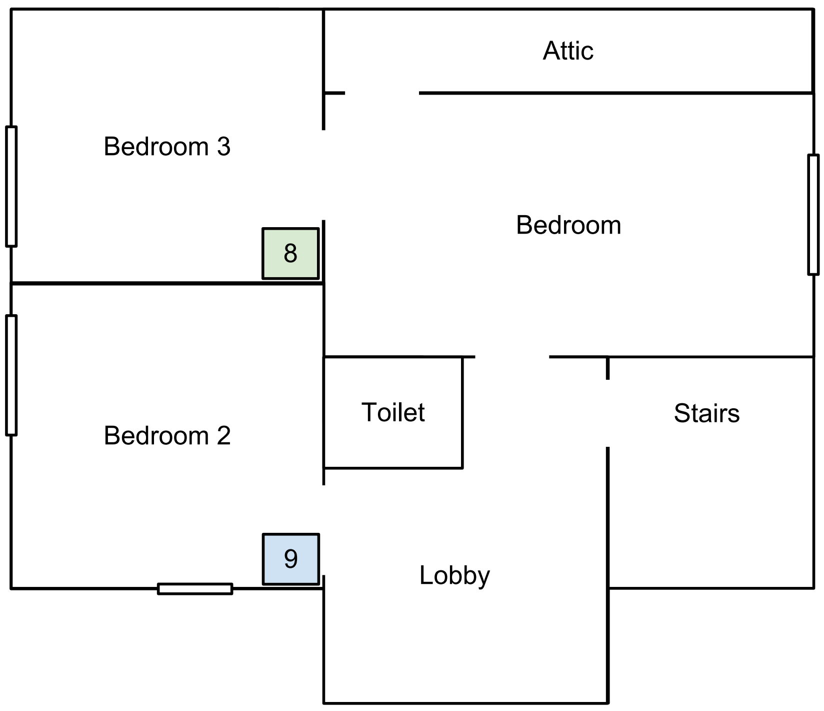

Figure 4.2: Layout of the second floor and the locations of the sensor units.

Example line: 1356998401;21.0253;0.0032; ...

Like the sensor file names, the measurement file names contain a house ID and a sensor group ID. Again, the header line specifies the columns found in the file. The first column is always an Unix timestamp which tells how many seconds have passed since 1.1.1970 until the the measurement was made. The rest of the columns specify the IDs and the types of the values the sensor is emitting. The numbers in pattern *ID_8_11_51* are house ID, sensor group ID and sensor ID, respectively. After that there is the variable type specified.

The complex events we are trying to detect are defined in Chapter 4.4.

4.2 Implementation

A test version of the predictive event processing network was created with Java programming language. Java was chosen because of several reasons:

- Java is the main development language of MMEA platform
- Esper CEP Engine provides easy-to-use API for Java
- Test data is available via a Java library
- The performance of Java is sufficient

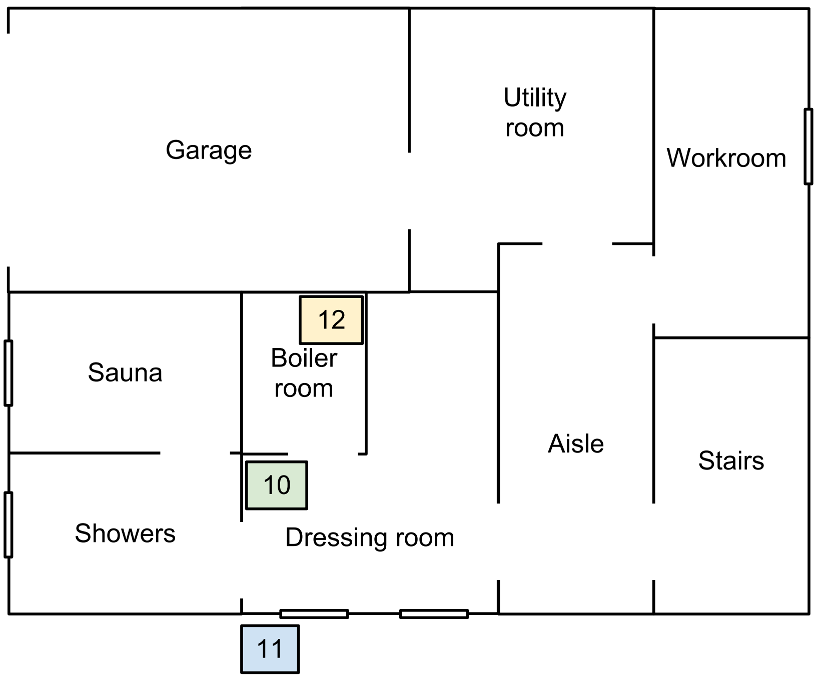

Figure 4.3: Layout of the basement and the locations of the sensor units.

Unlike the MMEA platform that uses PostgreSQL and Amazon S3 as database solutions, I have chosen MySQL as my development database because I am more familiar with it. I use an ORM (Object-relational mapping) layer called Hibernate between my program and the database. Thus, using a different database solution is only a matter of changing the database adapter from MySQL to something else in the Hibernate configuration file.

The goal of this chapter is to present a framework for integrating a predictive component alongside with a Event Processing Network. The approach is quite technical meaning that anyone with adequate skills in Java and Esper could replicate the system and improve it with their own expertise.

4.2.1 Data Structures

The test data is mapped into four Java classes that directly correspond to tables in my own MySQL database. This mapping is done with Hibernate library, which allows the programmer to annotate, that is, add table specifications Java classes. Then, Hibernate creates the database schema using these annotations. In the following I will describe these four classes in a compact form stating only the name of the class and parameter types and names.

First class represents the house we are dealing with:

```
public class House {
        int houseId;
        String name;
        Map<Integer, Sensor> sensors;
        Set<MeasurementUnit> units;
},
```

where, the parameter *sensors* is a map from sensor IDs to sensor objects and the parameter *units* contains all the measurement events from that house.

The second class represents a single sensor:

```
public class Sensor {
        House house;
        int sensorId;
        String name;
}
```

The third class represents a single measurement event which, in turn, may contain multiple measurement values from different sensors:

```
public class MeasurementUnit implements TimedEvent {
        House house;
        Date timestamp;
        Set<MeasurementValue> values;
}
```

The last class represents a single measurement value:

```
public class MeasurementValue {
        Sensor sensor;
        MeasurementUnit measurementUnit;
        double value;
}
```

As can be seen, *MeasurementUnit* class implements an interface called *TimedEvent* which is a common interface for all events in this thesis. It requires the class to have a timestamp property that can be used to mark the occurrence of an event in the time space. Objects from the class *MeasurementUnit* are the most important ones because the event source emits them to the CEP engine. The processing of these events is described in the following chapter.

4.2.2 Predictive Event Processing Network

The Predictive Event Processing Network (PEPN) is a separate network as described in Chapter 3.2. In this thesis I have made the separation so clear that the two networks even have their own Esper engines. In some publications (e.g. [21]) the PEPN just integrates into the EPN's engine but in this thesis' approach some Event Processing Agents (EPAs) connect to both networks and transfer data between them.

The structure of the Event Processing Network (EPN) defines which Complex Events the system captures and it depends on the phenomenon in question. However, the structure of the PEPN is completely independent of the EPN meaning that the framework presented here should, in theory, work for each Complex Event type that is predictable with the predictive models used here. The phenomenon-specific EPNs (and its EPAs) will be defined in experimental design chapter. The framework for PEPN is presented in Figure 4.4.

On the left side of the figure there is the EPN that attaches to the event sources, processes the data and outputs a Primary Complex Event (PCE) to the event sink. In this concept we don't place any restrictions for neither the format of input data nor the structure of the data processing. However, the output of the EPN must be unique, that is, only one type of output is allowed. Thus, the output can be formalized as the following Java class

```
public class ComplexEvent {
        Date timestamp;
        String message;
},
```

where the timestamp marks the time when the complex event takes place and the message contains relevant information about the event type that can be shown to the user when the event happens.

In Esper an event listener specifies an EPL and implements the interface

```
public interface UpdateListener {
        public void update(EventBean[] newEvents, EventBean[] oldEvents);
}
```

When the EPL is triggered, the *update* method is called and the events entering the window are given in *newEvents* array while the events leaving the window can be found from *oldEvents*. [16] In this thesis, an Event Processing Agent (EPA) is required to provide an EPL clause and to implement the interface above.

The Predictive Event Processing Network (PEPN) has four different Event EPAs that collect and preprocess the incoming data.

CollectorEpa listens to EPN's event sources and collects predictor vectors for training and predicting phases. Its EPL is of the form

```
SELECT
        value('<sensorId>').value as value,
        time
FROM
        MeasurementUnit.win:time(<windowLength>)
OUTPUT SNAPSHOT
        every <windowDifference>
```

Here the first select item uses *MeasurementUnit*'s getter for measurement values and then uses its getter for value. The clause defines a sliding window whose length is given as a parameter *windowLength* for the EPL. The last part, *output snapshot every windowDifference*, triggers the *update* method with the given time interval. Then, Esper's pull API allows us to iterate through the window's contents. [16] This EPL creates the predictor vectors that are then consumed by other EPAs.

Next, as can been from Figure 4.4, *TrainingEpa* and *PredictingEpa* use the predictors emitted by *CollectorEpa* as their input. In this case, the predictors are instances of the class *MeasurementVector*, which is of the following form

```
public class MeasurementVector implements TimedEvent, LabeledSample {
        Date timestamp;
        Date endTimestamp;
        List<Double> values;
        Label label;
},
```

where values list contains the measurement values between *timestamp* and *endTimestamp*. *MeasurementVector* is a time series which we are trying to classify. It implements the interface *LabeledSample* which requires it to have a *Label* property. The *TrainingEpa* listens also to the EPN's *ComplexEvent*s and its EPL is as follows

```
SELECT predictor.*
        FROM pattern[
                every predictor=MeasurementVector
                -> (timer:interval(<waitingTime>)
                -> (timer:interval(<eventTime>) and <Negation>c=ComplexEvent))
        ],
```

which needs a bit clarification. The pattern in the EPL first looks for a *MeasurementVector* event from *CollectorEPA*. The *every* keyword initiates this lookup on each event. When an event is found, the first timer waits an interval of length *waitingTime*. Then, the second interval looks for an interval of length *eventTime* during which a *ComplexEvent* is found. The variable *Negation* is either empty or "not ", which allows us to detect both positive and negative instances. For this purpose, we can define two *TrainingEPA*s: one for positive instances with *Negation* being empty and one for negative instances with *Negation* set to "not ".

Now the *TrainingEPA* collects the predictors but the next step is to decide when to perform the actual training process. For this purpose a *TrainingInvokerEPA*, which is not shown in the pictures as it "lives" inside the *TrainingEPA*. It declares an EPL of the form

```
SELECT
        *
FROM
        pattern[every timer:interval(<trainingInterval>)],
```

which is triggered automatically based on the time variable *trainingInterval*. Then, the *update* is called and it executes the training of the model.

Like the *TrainingEpa*s, *Predictor*EPA is fed with predictors (*MeasurementVectors*) as shown in Figure 4.4. Both *TrainingEPA*s and *PredictorEPA* communicate directly with a specific model class that extends an abstract *Model* class. Sub-classes of Model implement a certain model, in this case either a DTW and kNN-based one or a Wavelet and SVM-based one. The Model collects labeled predictors from *TrainingEPA*s until its *train()* method is called.

The *PredictorEpa*, in turn, calls the Model's classify(*MeasurementVector*)-method, which returns the label for the given predictor. If the instance is classified as positive, a notification event is sent to the PEPN, which is then captured by *AlertingEPA*. Then, the *AlertingEPA* builds a *ComplexEvent* and outputs it to the original EPN. In this way, the PEPN creates a warning for the EPN.

4.2.3 Data Flow and Performance Tuning

The event source reads a block of data from the database and sends it to both EPN and PEPN as can be seen in Figure 4.4. Then, the event processing networks process the data with their own EPAs. Without giving any thought to concurrency, this approach faces the problem of blocking as the event source does not start reading new data until the processing of old data is finished.

To solve this problem we use threads that are kind of lightweight processes. [42] Unlike actual processes, threads share the same memory space and are actually contained within one process. On a single-core processor the operating system can use time slicing feature to allow threads to execute code almost concurrently. On a multi-core processor the system's capability for concurrent processes enhances significantly.

Our system has four threads: main thread, event source thread, EPN thread and PEPN thread. The last three threads are initialized and started from the main thread. To exchange data between two threads we need a data structure that supports concurrent processing. The data structure should be some kind of queue as the event source fills it with data and the two networks read from it. The Java programming language offers several implementations for concurrent queues. A suitable choice is *LinkedBlockingQueue*, a First-In-First-Out (FIFO) queue, which offers the following methods [42]

void put(Object) Inserts the specified element into this queue, waiting if necessary for space to become available.

void take() Retrieves and removes the head of this queue, waiting if necessary until an element becomes available.

In our system there are two of these queues: one between event source and EPN and one between event source and PEPN. Now the event source can read a batch of data from the database and call *put(e)* for each element. Then, both networks, being in an infinite loop calling *take()*, start processing data as soon as it becomes available.

4.2.4 Machine Learning Libraries

In this thesis I use two ready-made machine learning libraries: Java Machine Learning Library (Java-ML) [1] and Weka 3: Data Mining Software in Java [25]. This approach allows me to concentrate not on the implementation of the algorithms but on the proper use of the methods. More importantly, the algorithms used in the libraries have already been optimized and tested properly by machine learning experts.

Java-ML contains an implementations for Dynamic Time Warping (DTW) and k-Nearest Neighbor (kNN) algorithms. Together with Java-ML's *CrossValidation* class, it is fairly easy to choose optimal parameter values. This algorithm, however, lacks the possibility of receiving intermediate results. For this reason I created an implementation by following the pseudo-code from section 3.6.1.

For a given set of parameters, the cross validation algorithm returns an instance of the class *PerformanceMeasure*, which contains the variables TP, FP, TN and FN. From these I calculate the value for the AC_d as described in (109)-(116). Then, the set of parameters with the lowest AC_d is chosen.

Java-ML has a wrapper for LibSVM which is an efficient Support Vector Machine library [7]. The interface provides Java classes that call the fast underlying C implementations. LibSVM comes with a class called *GridSearch*, which performs the grid search for parameters C and γ as described in Chapter 3.5.7. Again, I implemented the grid search in order to receive intermediate results.

In this thesis the Weka library is used to perform the wavelet transform and it supports the Discrete Haar Wavelet Transform. The conversion between Weka and Java-ML data formats is easy as their *Instance* classes provide getter and setter method for plain double arrays that contain the vector data.

4.3 MMEA Platform Architechture

4.3.1 Technology Overview

MMEA platform is designed to use Service Oriented Architecture (SOA) which means that different processes and components are implemented as independent and flexible services. These loosely coupled services use one another through open interfaces. This architecture pattern facilitates the integration of new components into the system and allows the services to be distributed into several physical machines.

The MMEA platform runs on the Amazon Web Services (AWS) cloud which consists of Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3). The former is a cloud computing platform where users can install virtual servers called instances and the latter is a cloud storage service where users can store application data through web services. Both services are billed as a pay-as-you-go service which means that users can scale the computing resources whenever needed and pay only for the resources used. By using a cloud-based approach the costs and the amount of maintenance of physical servers can be eliminated.

The SOA principle in MMEA is implemented using an open-source Enterprise Service Bus (ESB) from WSO2 software company. WSO2 ESB is a lightweight and high performance integration layer between different services. It uses Java Message Service (JMS) for exchanging messages between distributed and loosely-coupled services. The ESB integrates the five other MMEA layers

Sensor Layer Connects to sensor systems with sensor-specific adapters

Storage Layer Stores data into a database and loads data from it

Model Layer Runs computations based on problem-specific models

Control Layer Provides interfaces for monitoring and managing the system

Presentation Layer Provides data for external applications

These layers are shown in Figure 4.5.

At the core of the application layer there is a component called publish/subscribe which works over SOAP, an XML-based web service for messaging. External applications can publish their data feed and send their new data into the MMEA platform. Other parties can subscribe to these feeds and utilize the data in their processes, such as alarm services.

The MMEA platform has its own data format called MMEA Bus Message, which is an XML-based message format. It has a separate XML schema for common platform messages, sensor observation messages, forecast messages and complex event messages. For example, a sensor observation message contains information about the producer and the location of the sensor, as well as the measurand and the observed value. The sensor layer adapters define XSLT (Extensible Stylesheet Language Transformations) transformations from the producers' data formats into the MMEA Message Bus format.

4.3.2 Integrating the Predictive Component

The predictive complex event processing network must be integrated to the existing platform architecture so that the platform remains easily maintainable. As can be seen from the layer descriptions in the previous chapter, the MMEA platform has a model layer, which is used to run computations on problem-specific models. The predictive component can be integrated to this layer and run on the same physical computational cluster.

Another, and perhaps a better way to integrate the predictive component is to create an external application that is loosely coupled with the MMEA platform. In this approach the CEP engine and the predictive component are running on separate machine and use the publish/subscribe method provided by MMEA platform. Predictive component subscribes to feed that provides new sensor data. Then, having analyzed the data, it can publish the complex events, that is, the alarms back to the MMEA platform.

4.4 Experimental Design

In this section I first define the complex events that are predicted in the experimental part. Then, I describe some test settings that are employed in this thesis.

4.4.1 Complex Event Definitions

A simple Complex Event is a sensor value exceeding a certain predefined limit for a certain time. For each sensor type (temperature, CO_2, etc.) we can write a Java classes

```
public class ValueClassification {
        Set<Sensor> sensors;
        List<ValueClassificationLimit> limits;
}
```

and

```
public class ValueClassificationLimit {
        String name;
        double limit;
        int level;
}
```

The former represents a classification scale for one type of sensors and the latter is an actual limit. The *limit* attribute is the lower bound limit for an measurement value to belong to that class. The *level* attribute gets integer values beginning from 0 and can be used to display only *ClassificationEvent*s beginning from a given level. It can be thought as severity of the event.

Now we can further define a complex event that rises from exceeding a classification limit

```
public class ClassificationEvent extends ComplexEvent {
        Date timestamp;
        Sensor sensor;
        ValueClassificationLimit limit;
        String length;
},
```

which is a sub-class of *ComplexEvent*. It contains all the relevant information about the event: the timestamp of the event, the sensor that caused the event, the limit value which was exceeded and the length of time window that triggered the event.

An Event Processing Agent (EPA) called *ClassificationEPA* loads the classification limits from the database and builds the EPL clauses that detect the classification events. Let's say we have a sensor with *SensorID* 50 and the complex event happens when the measurement value exceeds the value of 100 for 20 minutes. Then, the following EPL detects the classification event of level 2 and builds a complex event that is sent to the CEP engine

```
INSERT INTO
        ClassificationEvent
SELECT
        unit.timestamp AS time,
        unit.value('50').sensor AS value,
        2 AS minLevel,
        '20 min' AS windowLength
FROM
        pattern[
                every (timer:interval(20 min)
                and unit=MeasurementUnit(value('50').value > 100))
        ]
```

Next, we need to specify the limits for each measurand. For CO_2 and VOC (Volatile Organic Compounds) we can use the indoor air quality classes defined by the Finnish Indoor Air Society. [49] The classification supports the evaluation of construction and renovation processes from the air quality's point of view.

Indoor Air Society has defined three classes, S1, S2 and S3, which describe the indoor air quality in the following way

S1: Individual Excellent indoor air quality. No detectable odors or sources of pollutants. Controllable temperatures with no overheating.

S2: Good Good indoor air quality. No disruptive odors or sources of pollutants. No detectable breeze but overheating possible during summer.

S3: Satisfactory Air quality and temperature conditions fulfill construction requirements.

For CO_2 and VOC the class limits are shown in Table 4.4. For VOC the corresponding values from our sensor are shown in parentheses.

Table 4.4: Air quality classifications for CO_2 and VOC.

Class	CO_2 (ppm)	VOC ($\mu g/m^3$)
S1	0-700	0-200 (0-10)
S2	700-900	200-300 (10-20)
S3	900-1200	300-600 (20-30)
S4	>1200	>600 (>30)

The particle detector counts the number of small particles in two size groups: particles with diameter between 1 μm and 5 μm and particles with diameter greater than 5 μm. The former group consists of fine dust, smoke, mold and bacteria while the latter is made of coarse dust, pollen and dust mite casings. The detector in our test house, Dylos DC1100, is calibrated to detect particles half the size mentioned before, that is 0.5 μm and 2.5 μm.

On the backside of the particle detector device, there is an air quality chart defined for the smaller particle size. This classification is presented in Table 4.6. The first column defines the classification in our system. The other two columns have been read from the device's backside. In the range 0-300 three classes, excellent, very good and good, have been combined into one class, PC1.

Table 4.5: Air quality classifications for CO_2 and VOC.

Class	Particle count	Description
PC1	0-300	Good/Excellent
PC2	300-1050	Fair
PC3	1050-3000	Poor
PC4	>3000	Very Poor

Similar classifications can be defined for heating and electricity consumption by determining the class ranges from the historical data. In more detail, the ranges were chosen to cover most often occurring peak values during the turning of the season. In this way, the most relevant complex events could be created for each time period (time of year). The classes are presented in table 4.6

Table 4.6: Classifications for electricity and heating.

Electricity Class	Consumption (kW)	Heating Class	Consumption (kW)
EC1	0-2	HC1	0-10
EC2	2-3	HC2	10-20
EC3	4-5	HC3	20-30
EC4	>5	HC4	>30

4.4.2 Selecting Parameters and Running the Tests

Two different prediction schemes are tested. The first starts with a training period of *trainingDays* days and continues with a series of testing periods of length *testingDays* for the rest of the data. The latter first trains the model with *trainingDays* days and then runs *testingBatches* single tests of length *testingDays*. This is process is then repeated until the data ends.

A long data set may show radical changes in indoor air quality conditions. If that is the case, then the second testing scheme which trains the model more often should perform better. This is one of the main focus points of the experimental section.

Our data collection phase has five different constants which were introduced in the previous chapters. A clarification for the first three parameters was shown in Figure 2.

windowLength Length of measurement vector that is used as a predictor.

waitingInterval Part of the prediction horizon that is ignored.

eventInterval Part of the prediction horizon which determines the class of the predictor depending whether or not a complex event happens within it.

windowDifference Determines how often a new predictor is created. Difference between two predictors' start points.

exceedTime Determines the minimum time interval that triggers a classification event, that is, a sensor value exceeds a certain limit.

Depending on the sensor, a certain combination of these parameters works better. For each sensor, each parameter is given a set of possible values. Then, the tests are run for each combination of parameters and the best combination is selected. The possible values are selected by inspecting the time series graph.

In addition to these parameters, the prediction models have parameters of their own, too. Each training process is run as a grid search and the best combination of the model parameters is used in the testing phase. For DTW and kNN based model those parameters are

radius Maximum deviation from the optimal path in DTW (see Chapter 3.4.2). Possible values: 5 and 50.

k The only parameter for k-Nearest Neighbour algorithm (see Chapter 3.4.3). Possible values: 3, 7 and 15.

The selected possible values produce a total of $2 \cdot 3 = 6$ combinations for the grid search. The smaller the *radius* for DTW is, the less accurate the algorithm is because the allowed deviation from the optimal path is smaller. The value of k for kNN determines how sensitive the classifier is to noise; the larger the parameter is, the less sensitive the classifier is to noise [18].

For SVM the parameter grids were presented in Chapter 3.7.6.

γ Parameter for the Gaussian Kernel. $a_0 = 3$

C Soft margin parameter. $C_0 = 5$

The selected values produce a total of $7*6 = 42$ combinations for the grid search.

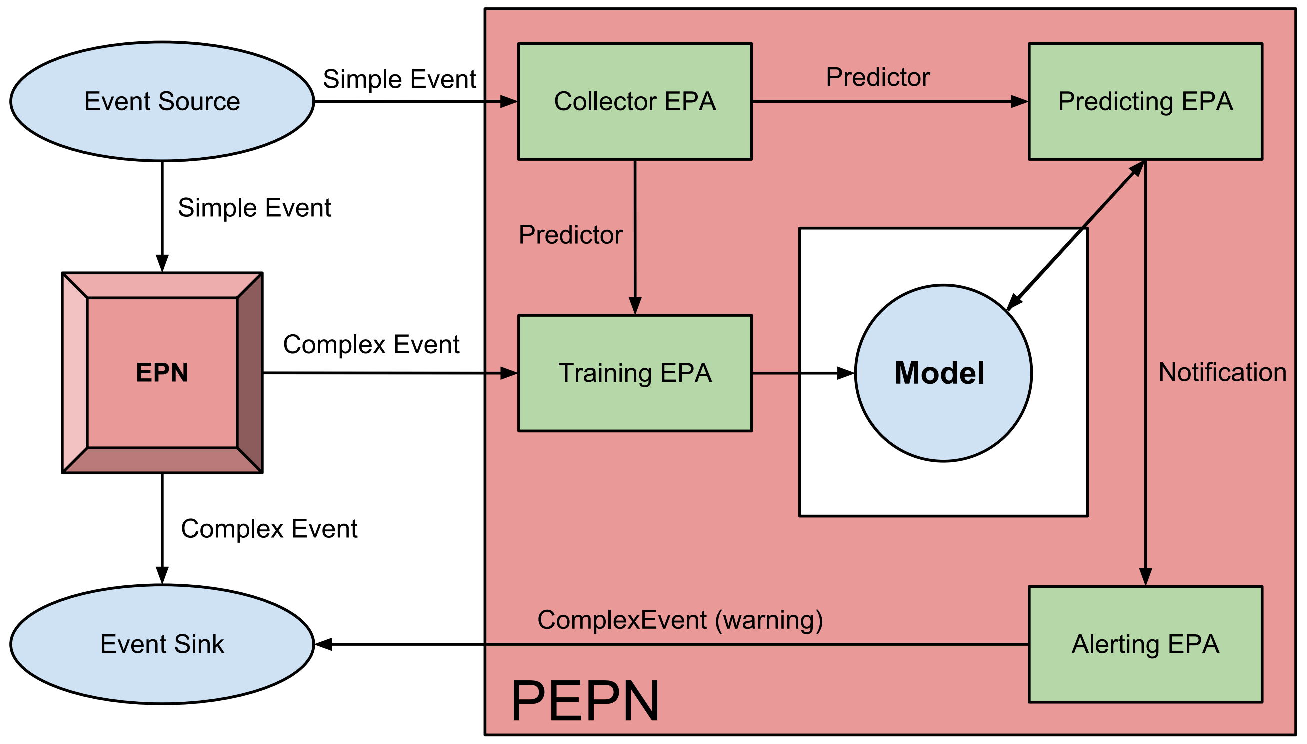

Figure 4.4: Event Processing Network (EPN) and Predictive Event Processing Network.

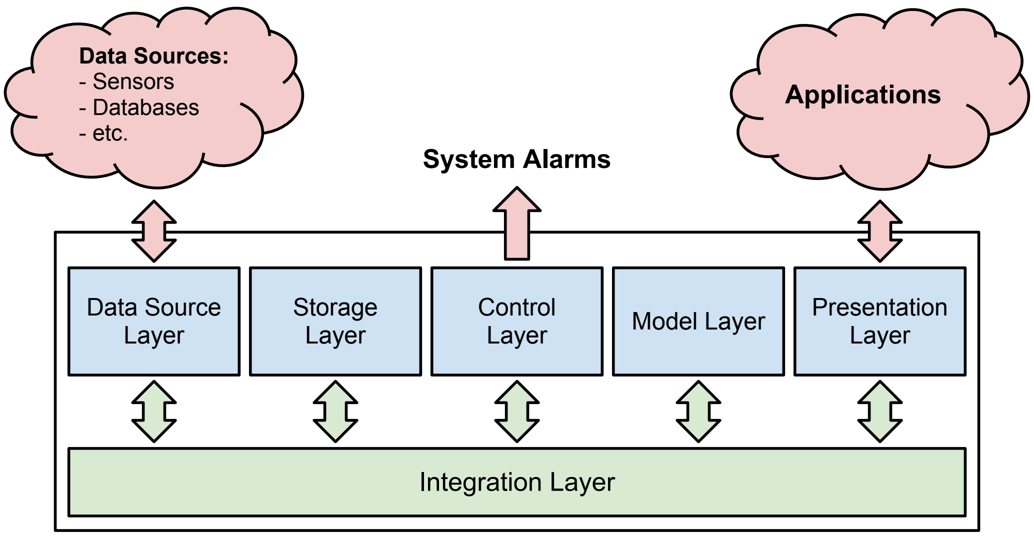

Figure 4.5: MMEA Platform layers.

Chapter 5

Results

5.1 Goodness Indicators

5.1.1 Classification With DTW and kNN

Variable: CO_2 Concentration

The first test is performed with CO_2 sensor from the bedroom 1 on the second floor (group 8 and sensor ID 646 in Table 2). The test data is from November 1st, 2012 until February 28th, 2013. One example week from that four month period is shown in Figure 5.1. The classification levels defined in Chapter 4.1 are shown in the figure as well as one missing part that is covered with linear interpolation. For CO_2 the minimum level for an alert is set to S2, which is the red line in the picture.

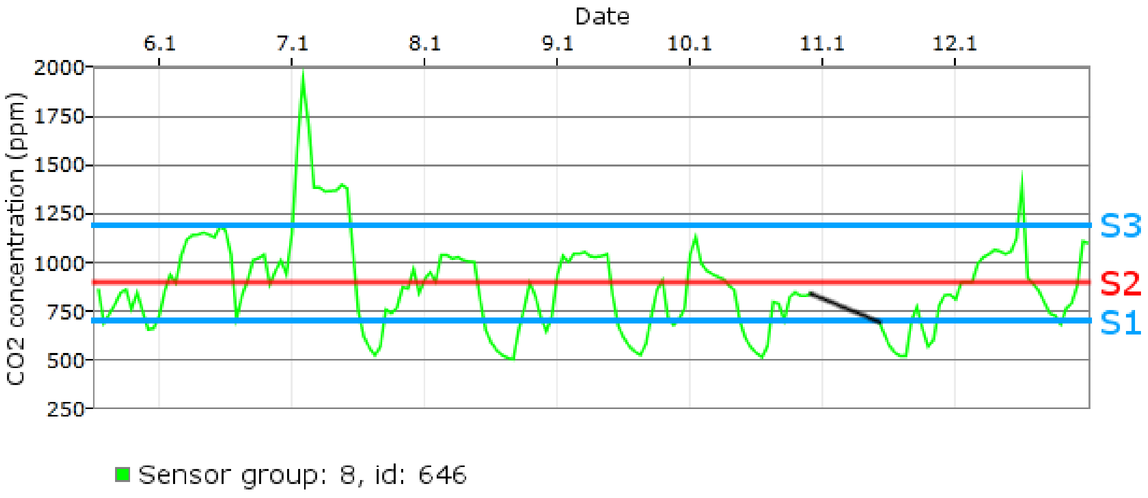

Figure 5.1: Example time series from CO_2 sensor. The S1, S2 and S3 mark the classification limits.

Table 5.1: Results for November 2012 with DTW and kNN model ($W = 0.5$). The definition of ROC distance AC_d is shown in (3.116). For each row the number of positive samples is P = TP + FN = 134 and the number of negative samples is N = FP = TN = 99.

radius	k	TP	FP	TN	FN	TPR	FPR	Accuracy	$\mathbf{AC_d}$
5	3	110	18	81	24	0.92	0.13	0.82	0.16
5	7	103	8	91	31	0.77	0.060	0.83	0.17
5	15	101	9	90	33	0.75	0.067	0.82	0.18
50	3	108	15	84	26	0.81	0.11	0.82	0.16
50	7	102	7	92	32	0.76	0.052	0.83	0.17
50	15	103	6	93	31	0.77	0.045	0.84	0.17

First, let us investigate the effects of DTW and kNN parameters on the classification results. It is reasonable to suppose that the values of *radius* and k should have only a minor effect on the classification results. The time parameters were set to the following values

- windowLength: 4 hours
- windowDifference: 1 hour
- waitingInterval: 30 min
- eventInterval: 2 hours
- exceedTime: 30 min

These values mean that a new four-hour predictor vector is created every hour. This predictor is considered positive if the sensor value exceeds the limit value for 30 minutes within two hours after the predictor and a waiting time of 30 minutes.

A 30-day period from November 2012 with a total of 267,524 events was used to test the six parameter combinations. The first 70 % of the data was used for training and the rest for testing. The classification results are shown in Table 5.1.

As can be seen from the results, the parameters *radius* and k do not have much effect on the classifier performance. Thus, the simplest possible model is selected. For the rest of the experiments values $radius = 5$ and $k = 3$ will be used.

Next, we perform the two experiments described in Chapter 4.4.2. The first test trains the model with 14 days of data and then runs tests in batches of 7 days. The second test runs three iterations, each beginning with a training period of 14 days and then three 7-day test batches. The parameters are the same as in the previous test except for the *exceedTime* which is now set to 1 hour.

The results from these experiments are shown in Table 5.2. As can be seen from the table, there is no significant difference between constantly updating model and using the the same model for all tests.

The CO_2 consumption seems to be quite easy to predict with this data set because the peaks are quite wide and they occur with constant intervals. The EPN

Table 5.2: DTW and kNN based model. 1: Periodically trained model. 2: Model that is trained only once.

Week	**1: Purpose**	**1: Accuracy**	**2: Purpose**	**2: Accuracy**
1	Training	-	Training	-
2	Training	-	Training	-
3	Testing	0.72	Testing	0.72
4	Testing	0.81	Testing	0.81
5	Testing	0.71	Testing	0.71
6	Training	-	Testing	0.75
7	Training	-	Testing	0.86
8	Testing	0.88	Testing	0.94
9	Testing	0.91	Testing	0.94
10	Testing	0.75	Testing	0.75
11	Training	-	Testing	0.84
12	Training	-	Testing	0.86
13	Testing	0.76	Testing	0.72
14	Testing	0.81	Testing	0.83
15	Testing	0.81	Testing	0.84

Table 5.3: Performance of DTW and kNN based model in the spring of 2013. The mechanical ventilation system was installed in the beginning of April.

Start	**End**	**P**	**N**	**TP**	**FP**	**TN**	**FN**	**TPR**	**TNR**	**Accuracy**
01.03.13	08.03.13	42	61	19	1	60	23	0.45	0.98	0.77
08.03.13	15.03.13	17	70	7	1	69	10	0.41	0.99	0.87
15.03.13	22.03.13	11	76	2	2	74	9	0.18	0.97	0.87
29.03.13	06.04.13	32	57	25	0	57	7	0.78	1.00	0.92
06.04.13	13.04.13	2	83	0	3	80	2	0.00	0.96	0.94
13.04.13	20.04.13	2	83	0	2	81	2	0.00	0.98	0.95
20.04.13	27.04.13	2	83	1	1	82	1	0.50	0.99	0.98

is likely to capture two kinds of complex events: ones that actually precede peaks and ones that are within a peak indicating that the peak will continue.

As of the beginning of April, 2013, the test house was equipped with a mechanical ventilation system which improved the air quality significantly. This change eliminates the periodic increase in CO_2 levels we tried to predict in the previous tests. As a comparison, the model was evaluated with four one-week periods from March and April, 2013. These results are shown in Table 5.3.

As can be seen from the number of positive testing samples in Table 5.3, the CO_2 levels decreased after the installation of mechanical ventilation system. Or at least the periodic peaks disappeared. The model now struggles with positive sample with *TPR* being much worse than in the previous tests. In the winter the average *TPR* was 0.76 for the same model.

Variable: VOC

As a second test variable we are using Volatile Organic Compounds (VOC). The sensor ID is 789 and the group is 7. It is located in a bedroom in the first floor.

As the last chapter indicated, we would not get much advantage from parameter optimization. Thus, we continue using the same values $k = 3$ and $radius = 5$. A sample week of the data is shown in Figure 5.2.

As the figure shows, this time the peaks that exceed the limit value (S1) are much sharper. Thus, we first try to find the optimal time constants. The test data is again from November, 2012.

Two time parameters, *windowDifference* and *exceedTime*, are kept as constants with values 4 hours and 5 min respectively. A small *exceedTime* is required to detect the sharp peaks. The prediction horizon parameters, *windowLength*, *waitingInterval* and *eventInterval* are varied. For each combination a three-fold cross validation is performed and average performances are calculated. The results are shown in Table 5.4.

The results show that the model misclassifies too many positive samples as negatives. The best parameter combinations are highlighted in the table. Clearly

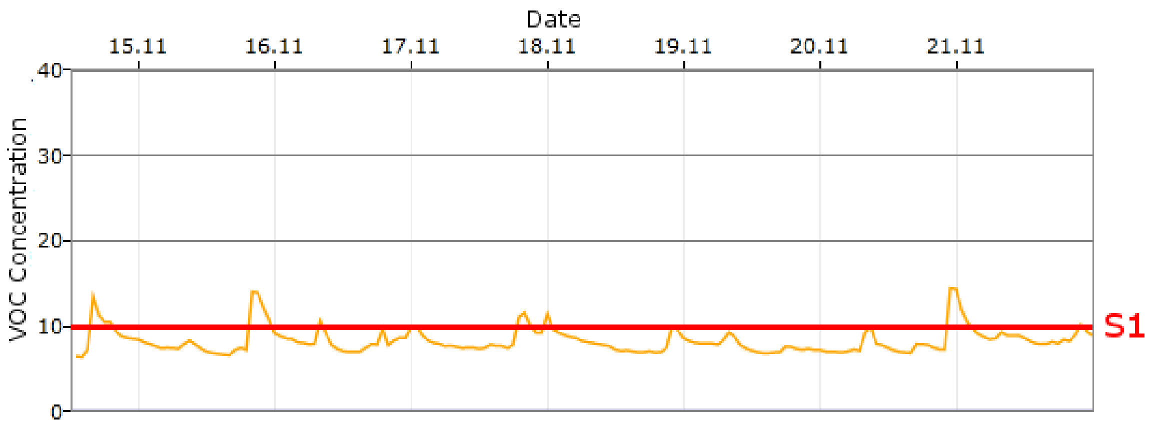

Figure 5.2: VOC concentration.

Table 5.4: Cross validation performances for DTW and kNN based model with different time parameters and VOC as measurand.

windowLength	**waitingInterval**	**eventInterval**	**TPR**	**FPR**	**Accuracy**	**AC$_d$**
2 hours	30 min	1 hour	0.54	0.08	0.77	0.57
4 hours			0.56	0.05	0.79	0.54
8 hours			0.38	0.09	0.71	0.68
2 hours	1 hour		0.45	0.12	0.70	0.65
4 hours			0.55	0.08	0.76	0.58
8 hours			0.34	0.14	0.64	0.71
2 hours	2 hours		0.43	0.13	0.68	0.66
4 hours			0.48	0.07	0.73	0.62
8 hours			0.33	0.21	0.60	0.73
2 hours	30 min	2 hours	0.52	0.13	0.72	0.61
4 hours			0.55	0.09	0.75	0.58
8 hours			0.38	0.18	0.63	0.70
2 hours	1 hour		0.45	0.13	0.69	0.65
4 hours			0.57	0.08	0.77	0.57
8 hours			0.35	0.21	0.60	0.72
2 hours	2 hours		0.43	0.13	0.67	0.66
4 hours			0.50	0.14	0.70	0.63
8 hours			0.34	0.22	0.59	0.73

windowLength of 4 hours works best. Other two parameters do not show that clear impact so we choose 30 min for *waitingInterval* and 1 hour for *eventInterval* because they produce a simpler model.

The selected model is tested with data starting from the beginning of December 2012. Figure 5.3 shows True Positive Rate (*TPR*) and False Positive Rate (*FPR*) for each week.

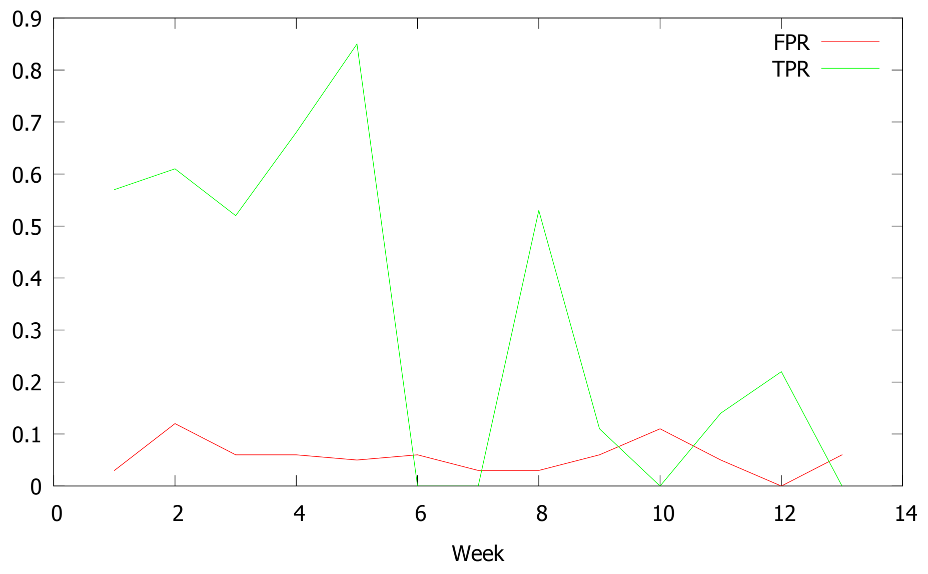

Figure 5.3: Weekly *TPR* and *FPR* for DTW and kNN based model starting from the beginning of December 2012.

The *FPR* is very small as it was in the parameter selection, too. The *TPR* maintains its level at 0.50-0.60 for five weeks and then drops significantly. Looking at the data shows that the number of limit-exceeding peaks almost disappear in the wintertime. Thus, it is very difficult for the model to predict these rare events.

5.1.2 Classification with Wavelets and SVMs

Variable: CO2

First, let us look at the product of the Haar Wavelet Transform. Figure 5.4 shows the original time series and Haar Wavelets with three different levels, 1, 2 and 10. The Weka Wavelet algorithm chooses the level from equation

$$\textbf{level} = ceil\left(\frac{\log length}{\log 2}\right), \tag{5.1}$$

where *length* is the length of the time series and *ceil()* rounds the answer up to the nearest integer. When the length of the time series is about 1000 points, the equation gives 10 levels.

This time the model parameters, γ and C should have a major impact on the classification results. Hence, we will perform an actual cross validation with 5 folds as described in Chapter 3.6.1. The data used is again from November 2012 so the resulting classifier can be compared to the one in the previous chapter.

For each parameter combination a cross validation is performed. Then, an average AC_d from Equation 3.116 is calculated for each parameter combination. The best parameter combination is then used for training the model.

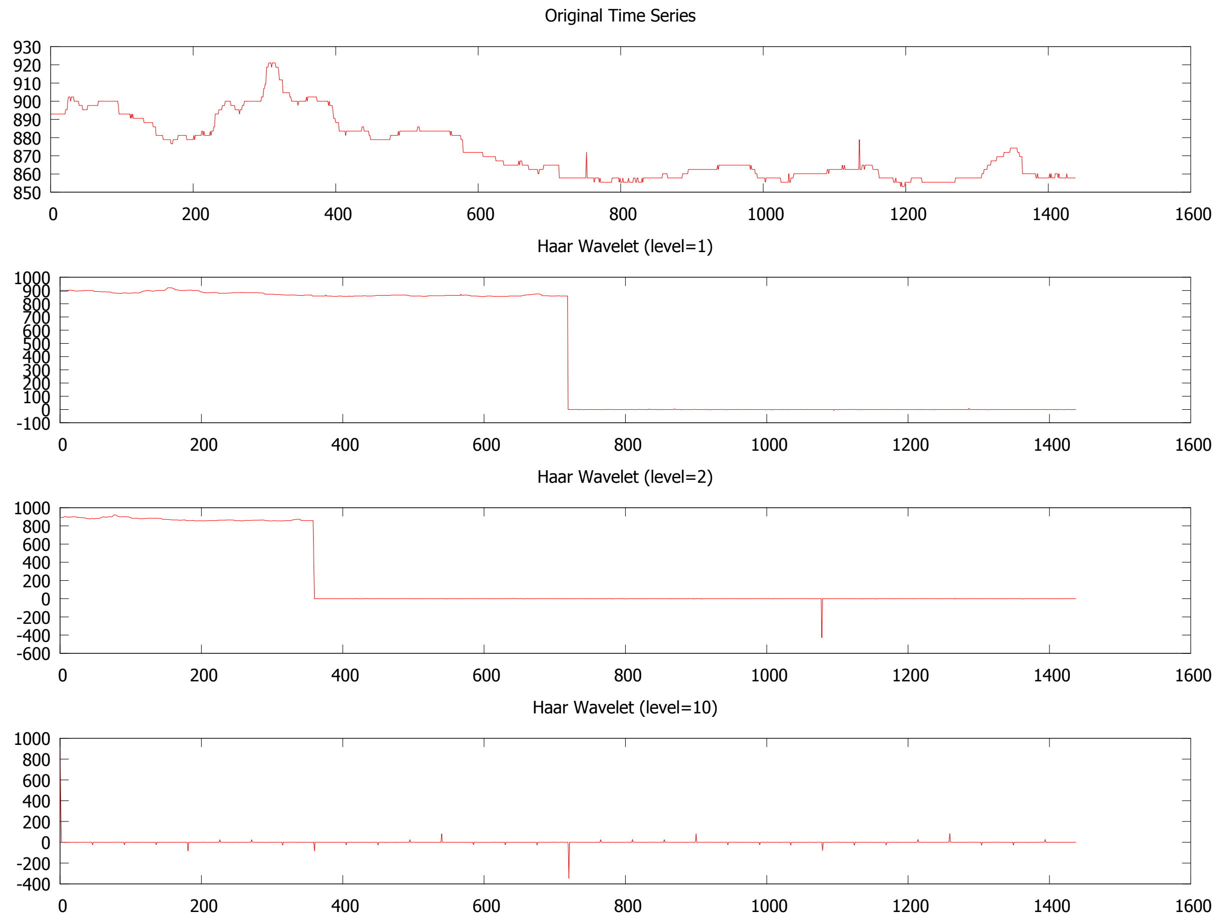

Figure 5.4: Original time series and Haar Wavelets with three different levels, 1, 2 and 10.

For parameter γ the grid values are calculated using the Jaakkola Heuristics described in Equation 3.105 with a_0 set to five. This results in 11 different values for γ. For parameter C we use a logarithmic grid $\{10^i\}$, $i = -2, ...6$ to first find out the correct magnitude for C.

The average accuracies and calculated AC_ds for C and γ are shown in Figures 5.5 and 5.6. Each point in the graphs is an average value of all the test runs with the given parameter value. There were a total of 55 runs for each value of C and 45 runs for each value of γ.

The average classifier performance improves as C increases to one. After that the accuracy decreases slowly. As the parameters might not be independent of each other, these graphs alone can not be used to select the optimal values. Figure 5.7 shows accuracy as a function of both C and γ. By inspecting the graphs a suitable combination for the parameters could be $C = 10$ and $\gamma = 0.05$.

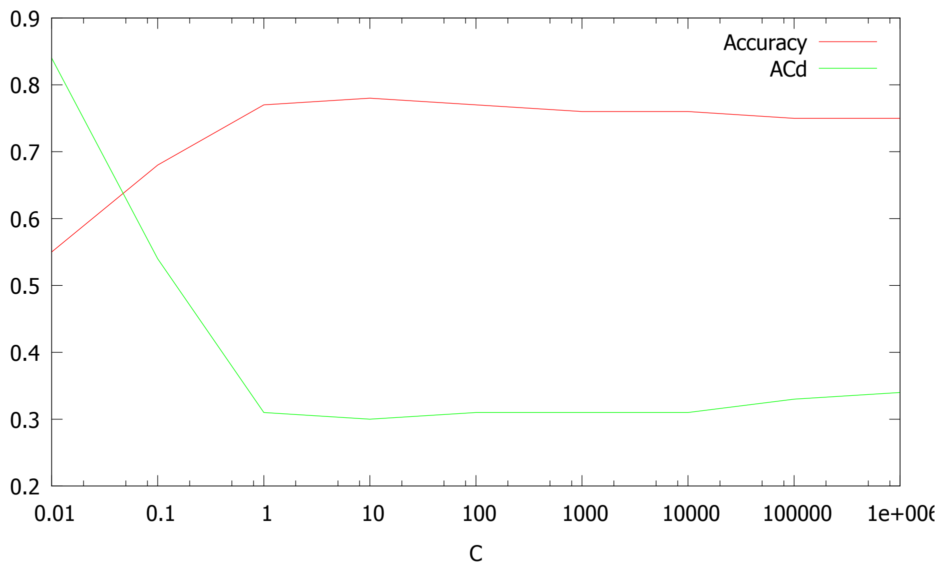

Figure 5.5: SVM classifier performance as a function of C.

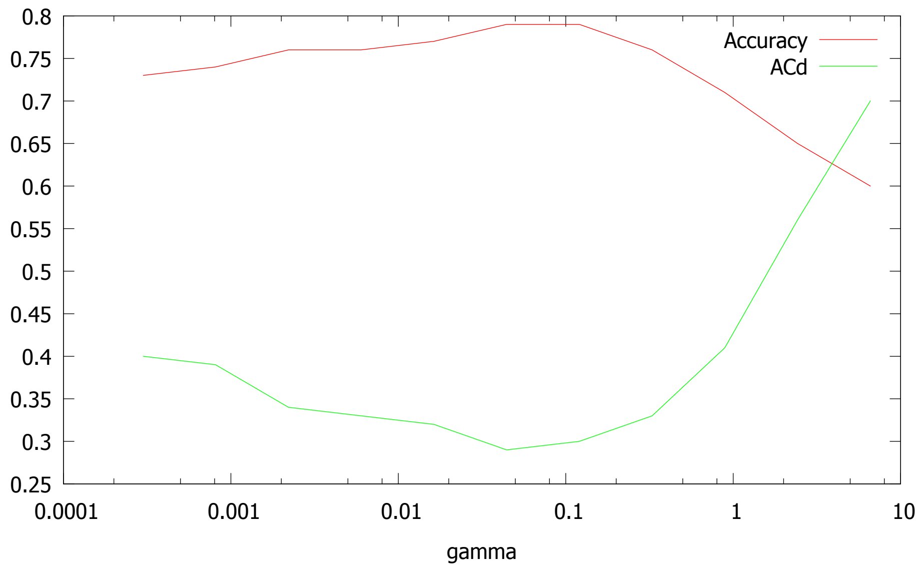

Figure 5.6: SVM classifier performance as a function of γ.

Next, the Wavelet and SVM model is trained with 14 days of data. This can not be done, however, with the same data we used for parameter selection because that approach would end up with overfitting. The values for TPR, FPR and Accuracy are shown in Figure 5.8.

The figure shows that the performance of this classifier varies much more than that of the DTW and kNN model.

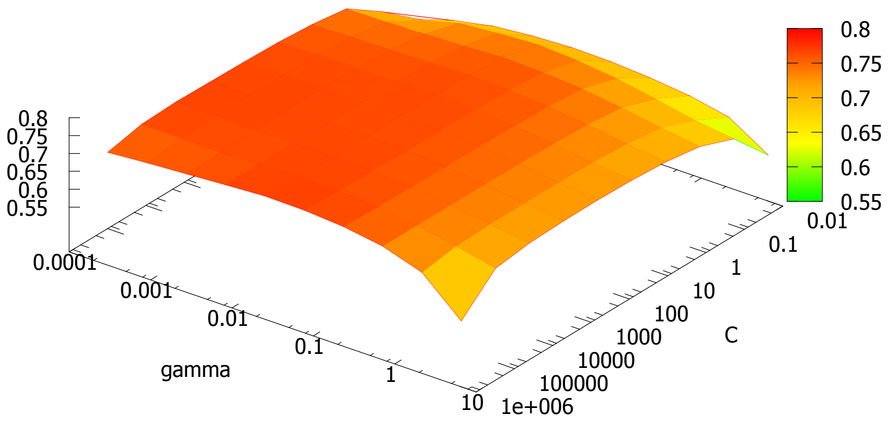

Figure 5.7: SVM classifier accuracy as a function of C and γ.

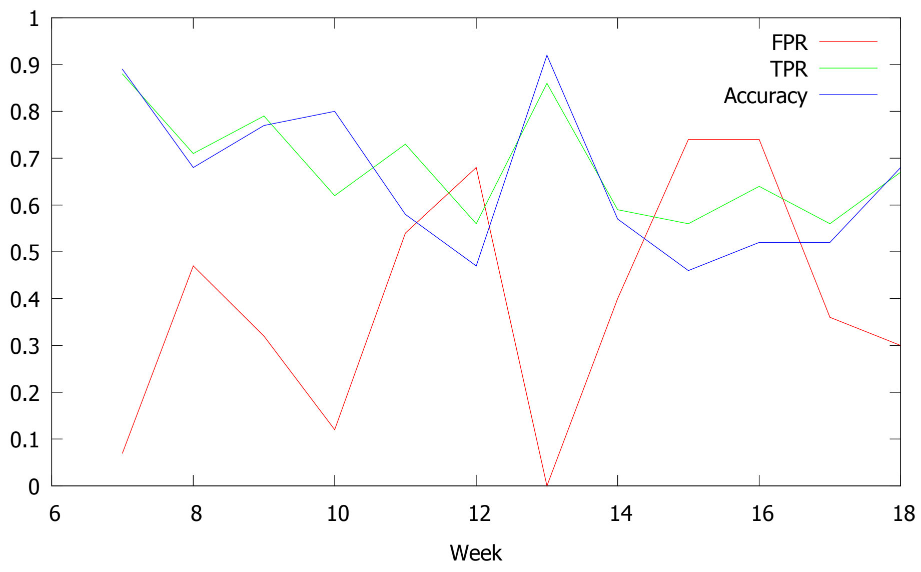

Figure 5.8: Wavelet and SVM based model performance with $C = 10$ and $\gamma = 0.05$.

5.2 Computational Performance

Standard DTW algorithm has time complexity of $\mathcal{O}(N^2)$, where N is the length of the time series. However, the FastDTW library we are using promises time complexity of $\mathcal{O}(N)$ [48]. The training phase of DTW and kNN is $\mathcal{O}(1)$ operation and takes only a few milliseconds because no calculations are performed. The given training set is simply saved into the model. Only when a new instance is classifier, DTW algorithm is run against each training sample and then k closest are selected for voting.

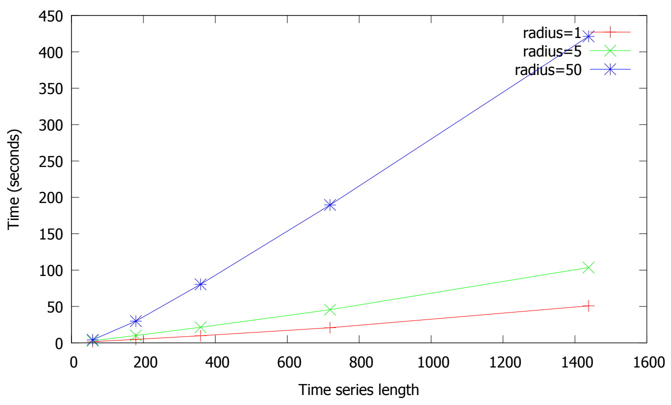

Figure 5.9: DTW and kNN based model's performance as a function of time series length.

Figure 5.9 shows the training time as a function of time series length. Different time series lengths were acquired by varying the *windowLength* parameter from 10 minutes to 4 hours. The smaller the *radius* parameter is, the shorter the calculation time is as the algorithm is stopped earlier. As the authors of FastDTW promised, the time complexity is linear for relatively small time series (< 2000 data points). For longer time series it soon becomes second-degree polynomial. [48]

The LibSVM promises time complexity of $\mathcal{O}(l)$, where l is the length of the time series if no kernel evaluations are required and $\mathcal{O}(nl)$ when n kernel evaluations are required [7]. To put it briefly, the SVM training should be linear.

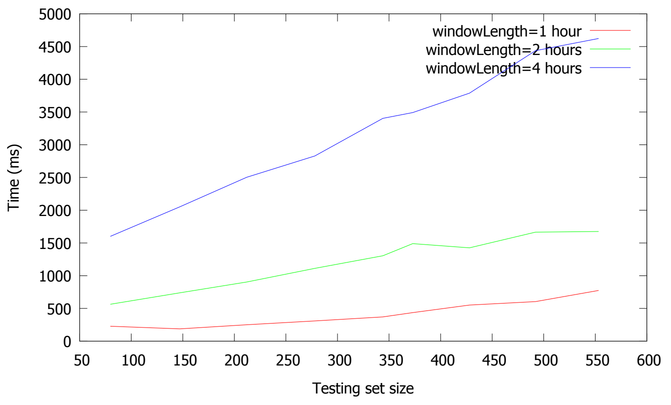

Figure 5.10: Wavelet and SVM based model's performance as a function of training set size.

Figure 5.10 shows the testing time of Wavelet and SVM based model when the training set size is held constant at 139 data points. Clearly, increasing the number of test instances increases the execution time linearly. This same applies for DTW and kNN based model.

If the number of test instances is also increased, the time complexity of DTW and kNN based model is no longer linear but rather follows a polynomial relationship, possibly a second-degree one. This is because each extra testing instance must be evaluated against each extra training set instance. The exact form of this polynomial could be found by regression analysis.

Wavelet and SVM based model, on the other hand, remains at linear computational time despite the increased number of both training and testing samples. This follows from the linear time requirements of both discrete Haar transform and

5.3 Discussion

The biggest difference between the two models was the computational time needed to train and test the models. DTW and kNN based model takes about few minutes while with Wavelet and SVM based model only a few seconds are enough.

Even though the FastDTW algorithm uses various methods, such as data abstraction and warping path constraints, to speed up the calculations, it still is a relatively slow algorithm. There is no way to decrease the number of DTW calculations as each testing instance must be evaluated against each training instance.

Calculating the Discrete Haar Wavelet Transform for small time series is not computationally demanding. It only consists of a series of addition and subtraction operations until the desired level is achieved. Weka Wavelet library doesn't provide any method to manually set to level up to which the transformation is performed. It would be useful to evaluate the classifier performance as a function level used for Haar Transform.

The core of LibSVM is written in C programming language which achieves better performance than if it was written in pure Java. LibSVM uses two techniques, shrinking and caching, to decrease the iteration time [7]. The shrinking method identifies and removes some bounded elements resulting in a smaller optimization problem. The quite advanced mathematics behind this method are not review here. The caching method stores kernel evaluations into memory for later use and thus avoids recalculations.

When it comes to classification performance, it was shown that regular peaks are much more predictable than ones that occur relatively rarely. With CO_2 the DTW and kNN based model achieved an acceptable performance with accuracy being over 80%, True Positive Rate (TPR) over 75% and False Positive Rate (FPR) under 10%.

When comparing a model that was trained once to a model that was trained with regular intervals, no differences were found. The reason for these findings might be that the signal was close to a stationary process whose mean and variance remained about the same for the whole testing period. It was also shown that the installation of mechanical ventilation system changed the indoor air conditions dramatically and the performance of the classifier plummeted.

With VOC variable it was shown that the system's time parameters have a major impact on the performance. For example, the optimal choice of *windowLength* parameter may increase the accuracy from about 60% to over 75%. The number of indoor air particles decreased in the winter and the classifier failed heavily after that.

The Wavelet and SVM based model is difficult to tune. In addition to system's time parameters, it has two parameters, C and γ that derive from the mathematical equations. With CO_2 it was shown that both these parameters have optimal values that maximize the accuracy and minimize the AC_d. However, the testing accuracy varied a lot compared to the first model. LibSVM provides additional parameters, such as ϵ for stopping the iteration and weight parameters for different classes, that are sure to affect the observed performance but were not investigated here.

The second model is much more sensitive to the available data. Even small changes in the time parameters transformed the classifier from a nearly perfect one to a completely useless one. Especially with VOC, which has sharp peaks occurring rarely, this model lacks some kind of dynamic time parameter tuning that would optimize the length and the difference of the time windows on the go.

All in all, the results were quite close to my expectations. The easy-to-use DTW and kNN based model outperformed the more sophisticated Wavelet and SVM based model. The latter model, however, showed potential for predicting regular peaks in less time then the first model. With a little bit more parameter tuning I am pretty confident that Haar Wavelets and SVMs can work together with reasonable results.

Chapter 6

Conclusions

The goal of this thesis was to develop a framework for predictive complex event processing. By combining CEP and Predictive Analytics (PA) the capabilities of CEP for pattern detection can be taken a step further towards a real-time alarm system. Building this kind of system consists of the following phases

1. Setting up the sensors
2. Enabling data flow into the CEP engine
3. Identifying the phenomena and describing them with EPL clauses
4. Building and configuring the predictive component to receive alarms
5. Taking necessary actions

The first phase, setting up the sensors, requires either setting up physical sensors or simulating ones with a computer software. Since there were data available from a real-life case, it was a easy choice to ask for a permission to use it. This choice, nevertheless, came with a potential risk of the data being uninteresting or too complicated. The two sensors, CO_2 and VOC were chosen for testing because of their large value range. The consumption variables, such as electricity and heating, had to be left out because their values were in pulses and not in their actual units. This conversion would have caused too much extra work.

The second phase, the data flow from the sensors into the CEP engine can be further divided into two phases: a data flow from the sensors into our predictive framework and a data flow inside our predictive framework. The former begins with wireless sensors emitting measurements to a server via a router. The latter consists of downloading the data from the server and formatting them to Java objects that can be inserted into the CEP engine.

At the core of the predictive framework is to Esper CEP engine which, as a state-of-the-art open-source solution, is gaining more and more popularity among CEP developers. In this thesis only a small fraction of Esper's capabilities were used. Only relatively simple EPLs with some temporal logic and timers were used. More advanced features, such as context-dependent reasoning and direct database connections, could be used to extend the predictive framework into a more comprehensive house automation software.

The third phase, writing the EPL clauses, requires domain experts to define meaningful complex events. Since I do not have enough experience in environmental issues, a very simple complex event type was chosen. Detecting a variable exceeding a certain limit is, however, an important one according to the Finnish Indoor Society. By their definition, the indoor air quality is solely determined by a set of limits for different variables. [49]

The next step from our "simple complex events" would be to detect and predict a combination of variables exceeding a limit. This would give a more comprehensive image of the living conditions. Then, additional independent variables, such as weekday and season, could be used to add cycle detection to the platform.

The fourth phase is the main focus point of this thesis. The predictive component is a separate CEP network that listens to the original network's inputs (measurements) and outputs (complex events) [21]. The novel idea of this thesis was to construct the predictors and their labels with CEP. All the previous work done in this area builds the training and testing set manually, or at least they have no mentions of using CEP for that.

The fifth phase requires domain expertise to define the necessary actions. When it comes to limit exceeding complex events that we are investigating in this thesis, a possible action could be opening a window. Other actions include adjusting heating, cooling or ventilation. In case of an emergency situation, such as rising carbon monoxide levels, a loud alarm should be played in order to alert the residents. Since this system is already connected to the outside world, also the authorities could be notified and called automatically in that case.

Taking necessary actions is closely tied with the time parameters in our predictive framework. The time parameters, *waitingTime* and *eventTime*, define the prediction horizon that begins with a warning interval. The warning interval corresponds to the time needed to prevent the complex event from happening. Should a single topic be chosen for additional research, it would definitely be a more thorough choice of time parameters.

This thesis has two different intersections with real-life applications. First, the motivation for this work stems from the MMEA research project which aims at creating a platform for environmental data exchange. The platform offers interfaces for external applications that can make use of the real-time sensor data. The predictive complex event processing platform that was developed in this thesis could be one of these applications.

Second, this thesis uses test data from ASTEKA project which builds intelligent houses that adapt to changing living conditions. Even though the real-life data is with no doubt very challenging to start with, the framework developed in this thesis showed some potential allowing smooth data flow and some promising result. Actually, the most successful parts of this thesis are related to data handling and integrating all the components together. With relatively little code actual sensor data is used for prediction and alerts are shown to the user.

The problems that were faced during the experimental part were mainly related to the applied methods and the available testing data. First, the used models, especially the Wavelet and SVM based one, have a fairly large number of parameters that affect the performance. For this reason each section in the results chapter is built from different types of results. This approach shows a cross-section of what kind of results can be achieved and how they can be represented in a tabular or a visual form. The parameter optimization needs definitely more investigation before the platform can be said to actually work.

Second, the available testing data was actual data from an actual test house. In real-life the CO_2 concentration is not the most interesting variable when it comes to alerting the residents. However, a CO_2 sensor located in a small room without mechanical ventilation is sensitive enough to show a lot of regular variation which our platform managed to predict very well. When the mechanical ventilation was installed and the regular peaks disappeared, the performance of our model decreased significantly.

As hypothesized before, the simpler DTW and kNN based model performed better than the more complex Wavelet and SVM model. Our complex event type, a variable exceeding a limit, can be predicted by detecting the rise in that variable. DTW captures this kind of information better probably because it stretches the two time series being compared so that the rising parts are matched. Wavelet transform, on one hand, is able to detect this kind of patterns but, on the other hand, lacks translation invariance, which means that the two signals might not be similar if they are in different phases. There are some methods available to overcome this problem. [55]

As already mentioned before, possible focus points for future research are

- Better evaluation of what kind of data and which variables can be predicted
- Defining more meaningful complex event with domain experts
- Optimizing model parameters more thoroughly

All of these were investigated to some extent but better results would definitely be obtained by focusing more on these three points. In the light of these shortcomings it is fair to say that the predictive framework developed in this thesis is just a proof-of-concept and still quite far from a real-life application.

Bibliography

[1] Abeel, T. & Van de Peer, Y. & Saeys, Y., *Java-ml: A machine learning library*, Journal of Machine Learning Research **10** (2009), 931–934.

[2] Adi, A. & Botzer, D. & Nechushtai, G. & Sharon, G., *Services computing workshops*, Services Computing Workshops, IEEE, 2006, SCW '06 Conference, pp. 7–12.

[3] Augusto, J. C. & Nugent, C. D. (ed.), *The use of temporal reasoning and management of complex events in smart homes*, 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, Valencia, Spain, 2004.

[4] Bellemans, T. & De Schutter, B. & De Moor, B., *Model predictive control for ramp metering of motorway traffic: A case study*, ESAT-SCD (2006), 406–411.

[5] Boswell, D., *Introduction to support vector machines*, Tech. report, Caltech Institute of Technology, Pasadena, CA, USA, 2002.

[6] Buytendijk, F. & Trepanier, L., *Predictive analytics: Bringing the tools to the data*, Tech. report, Oracle Corporation, Redwood Shores, CA, USA, 2010.

[7] C. Chang, C. & Lin, *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems & Technology **2** (2011), 27:1–27:27, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[8] Chen, C. & Huang, Y. & Li, G., *Hourly heating load prediction of radiant floor heating system based on the BP neural network*, Advanced Materials Research **243-249** (2011), 4913–4917.

[9] Chen, H. & Jiang, G. & Ungureanu, C. & Yoshihira, K., *Failure detection & localization in component based systems by online tracking*, ACM SIGKDD internation conference on Knowledge discovery in data mining, NEC Laboratories America, Inc., Princeton, NJ, USA, 2006.

[10] CLEEN Ltd, *Measurement, monitoring and environmental efficiency assessment - innovations through new thinking, MMEA factsheet*, http://www.cleen.fi/fi/Markkinointiviestint/MMEA_Factsheet_20120417.pdf, 2012.

[11] Dubin, R., *Theory building*, New York: The Free Press, New York, NY, USA, 1969, ISBN 9780029076200.

[12] DuChateau, P., *Basic facts about Hilbert space*, Tech. report, Colorado State University, Fort Collins, CO, United States, 2002.

[13] Dumbill, E., *Big data*, vol. 1, O'Reilly Media, Sebastopol, CA, USA, 2012, ISSN 2167-6461.

[14] Edwards, M. & Etzion, O. & Ibrahim, M. & Iyer, S. & Lalanne, H. & Moxey, C., *A conceptual model for event processing systems*, http://public.dhe.ibm.com/software/dw/webservices/ws-eventprocessing/ws-eventprocessing-pdf.pdf, 2010.

[15] Elkan, C., *Evaluating classifiers, lecture notes, winter 2012*, Tech. report, University of California, San Diego, CA, USA, 2012.

[16] EsperTech Inc., *Esper reference*, Version 4.6.0, http://esper.codehaus.org/esper-4.6.0/doc/reference/en-US/pdf/esper_reference.pdf.

[17] Etzion, O. & Niblett, P., *Event processing in action*, Manning Publications Co., Greenwich, CT, USA, 2010, ISBN 9781935182214.

[18] Everitt, B. S. & Landau, S. & Leese, M. & Stahl, D., *Miscellaneous clustering methods, in cluster analysis*, 5 ed., John Wiley & Sons, Ltd, Chichester, UK, 2011, ISBN 9780470977811.

[19] FICO, *What are the types of predictive analytics?*, http://dmblog.fico.com/2006/06/what_are_the_ty.html, 2006, Blog.

[20] Fong, K. M., *Time series forecasting using wavelet and support vector machine*, Master's thesis, Department of Mechanical Engineering, National University of Singapore, 2004.

[21] Fülöp, L. J. & Tóth, G. & Vidács, L. & Beszédes, á. & Demeter, H. & Farkas, L., *Predictive complex event processing: A conceptual framework for combining complex event processing & predictive analytics*, BCI '12 Proceedings of the Fifth Balkan Conference in Informatics (2012), 26–31.

[22] Gantz, J. & Reinsel, D., *Extracting value from chaos*, http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf, 2011.

[23] Gutierrez-Osuna, R., *L17: linear discriminant functions*, Tech. report, Texas AM University, College Station, TX, USA, 2011.

[24] Hai-Lam, B., *Survey & comparison of event query languages using practical examples*, Master's thesis, Insitut für Informatik, Ludwig-Maximilians-Universität, München, 2008.

[25] Hall, M., *The WEKA data mining software: An update*, ACM SIGKDD Explorations Newsletter (2009), 10–18.

[26] Hamilton, H., *Knowledge discovery in databases*, Computer Science 831: Lecture Notes, University of Regina, Canada.

[27] Hamilton, J .D., *Time series analysis*, Princeton University Press, Princeton, NJ, USA, 1994, ISBN 978-0691042893.

[28] Hastie, T., *The elements of statistical learning*, 2 ed., Springer, New York, NY, USA, 2009, ISBN 978-0387848570.

[29] Hautakangas, H. & Nieminen, J., *Anomaly detection using one-class SVM with wavelet packet decomposition*, Master's thesis, University of Jyväskylä, Department of Mathematical Information Technology, Jyväskylä, Finland, 2011.

[30] Jaakkola, T., *Using the Fisher kernel method to detect remote protein homologies*, ISMB-99, AAAI, 1999.

[31] Jordan, M. I., *Advanced topics in learning and decision making: The kernel trick*, Tech. report, University of California, Berkeley, CA, USA, 2004.

[32] Keogh, E. & Ratanamahatana, C. A., *Exact indexing of dynamic time warping*, Knowledge and Information Systems **7** (2005), 358–386.

[33] Kobielus, J., *Really happy in real time*, http://www.destinationcrm.com/Articles/Columns-Departments/Connect/Really-Happy-in-Real-Time-50530.aspx, 2008, Article.

[34] Kohonen, T. (ed.), *The self-organizing map*, Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1990, ISBN 3540679219.

[35] Kolehmainen, M., *Ympäristöinformatiikan asumisen energiatehokkuuteen ja terveellisyyteen liittyvät hankkeet, Lecture Notes*, 2010, University of Eastern Finland, Kuopio, Finland.

[36] Li, J., *Learning vector quantization & k-nearest neighbor*, Tech. report, Penn State University, University Park, PA, USA, 2008.

[37] Luckham, D. & Schulte, W., *Event processing glossary - version 1.1*, Hoboken, New Jersey: John Wiley & Sons, Inc., 2008, ISBN 978-0-470-53485-4.

[38] Marcelo, M. & Bizarro, P. & Marques, P., *A performance study of event processing systems*, Tech. report, CISUC, University of Coimbra, Portugal, 2009.

[39] Mitchell, T.M., *Machine learning*, McGraw-Hill, New York, NY, USA, 1997, ISBN 70070428077.

[40] Montgomery, D. C. & Peck, E. A. & Vining G. G., *Introduction to linear regression analysis*, Wiley, New York, NY, USA, 2012, ISBN 978-0-470-54281-1.

[41] Müller, M., *Information retrieval for music and motion*, ch. 4, Springer, New York, NY, USA, 2007, ISBN 978-3-540-74048-3.

[42] Oracle Inc., *Java platform, standard edition 6 API specification*, http://docs.oracle.com/javase/6/docs/api/, 2011.

[43] Pascual-Montano, A., *Dimensionality reduction*, Tech. report, Complutense University of Madrid, Madrid, Spain, 2007.

[44] Phillips, W. J., *Wavelets & filter banks course notes*, Tech. report, Dalhousie University, Halifax, Canada, 2003.

[45] Ramirez Pozo, A. T., *Measuring the performance of a classifier*, ch. 11, Universidade Federal do Parana, Curitiba, PR, Brazil, 2009.

[46] Ratanamahatana, C. A. & Lin, J. & Gunopulos, D. & Keogh, E. & Vlachos, M. & Das, G., *Mining time series data*, ch. 1, pp. 1069–1103, Springer, New York, NY, USA, 2005, ISBN 978-0-387-24435-8.

[47] Ruping, S., *SVM kernels for time series analysis*, LLWA 01 (2001), 43–50.

[48] Salvador, S. & Chan, P., *Toward accurate dynamic time warping in linear time and space*, 3rd Workshop on Mining Temporal & Sequential Data (2004), 561–580.

[49] Sateri, J., *Sisäilmastoluokitus 2008: Sisäympäristön uudet tavoitearvot*, Tech. report, Sisäilmayhdistys ry., 2008.

[50] Schölkopf, B. & Sung, K. & Burges, C. & Girosi, F. & Niyogi, P. & Poggio, T. & Vapnik, V., *Comparing support vector machines with gaussian kernels to radial basis function classifiers*, IEEE Transactions on Signal Processing **45** (1997), no. 11, 2758 – 2765.

[51] Shmueli, G., *Predictive analytics in information systems research*, Tech. report, Robert H. Smith School of Business, University of Maryland, College Park, MD, USA, 2010.

[52] Skön, J-P. & Rönkkö, M. & Raatikainen, M. & Juhola, M. & Främling, K. & Kolehmainen, M., *Research on computational intelligence for co-learning enabled healthy and sustainable housing*, University of Eastern Finland, Kuopio, Finland, Manuscript.

[53] Smith, J. O., *Mathematics of the discrete fourier transform (DFT), with audio applications*, 2 ed., W3K Publishing, 2007, ISBN 978-0974560748.

[54] Staelin, C., *Parameter selection for support vector machines*, Tech. report, HP Laboratories Israel, 2003.

[55] Struzik, Z. R., *The haar wavelet transform in the time series similarity paradigm*, Principles of Data Mining & Knowledge Discovery (1999), 12–22.

[56] Tappen, M., *Lecture 6 - linear classification*, Tech. report, University of Central Florida, Orlando, FL, USA, 2010.

[57] Theodoridis, S. & Koutroumbas, K., *Pattern recognition*, Academic Press, Waltham, MA, USA, 2009, ISBN 9781597492720.

[58] Verbunt, M. & Walser, A. & Gurtz, J. & Montani, A. & Schär, C., *Probabilistic flood forecasting with a limited-area ensemble prediction system: Selected case studies*, Journal of Hydrometeorology (2007), 897–909.

[59] Vincent, P., *The CEP market in 2011*, http://www.thetibcoblog.com/2011/03/04/the-cep-market-in-2011/, 2011, The TIBCO Blog.

[60] Waisberg, I., *Fourier & beyond: The wavelet transform*, Tech. report, Stanford University, Stanford, CA, USA, 2011.

[61] Wang, F. & Liu, S. & Liu, P. & Bai, Y., *Bridging physical & virtual worlds: Complex event processing for RFID data streams*, Lecture Notes on Computer Science, vol. 3896, 2006, Advances in Database Technology, pp. 588–607.

[62] Weiss, G. M. & Hirsh, H., *Event prediction: Learning from ambiguous examples*, Tech. report, Rutgers University, Department of Computer Science, 1998.

[63] Wenjian, W. & Liang, M., *An estimation of the optimal Gaussian kernel parameter for support vector classification*, ISNN (1), Springer, New York, NY, USA, 2008, ISBN 978-3-540-87732-5, pp. 627–635.

[64] Xing, Z. & Pei, J. & Keogh, E., *A brief survey on sequence classification*, ACM SIGKDD Explorations Newsletter **12** (2010), 40–48.

[65] Yeganeh, B. & Shafie Pour Motlaghb, M. & Rashidib, Y. & Kamalanc, H., *Prediction of CO concentrations based on a hybrid partial least square and support vector machine model*, Atmospheric Environment **55** (2012), 357–365.

[66] Yu, L. & Liu, H., *Feature selection for high-dimensional data: A fast correlation-based filter solution*, Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington, DC, USA, 2003.

[67] Zhang, L. & Zhou, W. & Jiao, L., *Wavelet support vector machine*, Cybernetics **34** (2004), no. 1, 34–39.