BeatDB: An end-to-end approach to unveil saliencies from massive signal data sets

by

Franck Dernoncourt
Master of Science, ENS Ulm, Paris (2011)
Master of Science, HEC, Paris (2011)
Master of Science, CNAM, Paris (2011)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2015

Author .....
Department of Electrical Engineering and Computer Science
December 9, 2014

Certified by .....
Una-May O'Reilly
Principal Research Scientist, CSAIL
Thesis Supervisor

Certified by .....
Kalyan Veeramachaneni
Research Scientist, CSAIL
Thesis Supervisor

Accepted by .....
Professor Leslie A. Kolodziejski
Chair, Committee on Graduate Students
Department of Electrical Engineering and Computer Science

BeatDB: An end-to-end approach to unveil saliencies from massive signal data sets

by

Franck Dernoncourt

Submitted to the Department of Electrical Engineering and Computer Science
on December 9, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Prediction studies on physiological signals are time-consuming: a typical study, even with a modest number of patients, usually takes from 6 to 12 months. In response we design a large-scale machine learning and analytics framework, BeatDB, to scale and speed up mining knowledge from waveforms.

BeatDB radically shrinks the time an investigation takes by:

- supporting fast, flexible investigations by offering a multi-level parameterization, allowing the user to define the condition to predict, the features, and many other investigation parameters.
- precomputing beat-level features that are likely to be frequently used while computing on-the-fly less used features and statistical aggregates.

In this thesis, we present BeatDB and demonstrate how it supports flexible investigations on the entire set of arterial blood pressure data in the MIMIC II Waveform Database, which contains over 5000 patients and 1 billion of blood pressure beats. We focus on the usefulness of wavelets as features in the context of blood pressure prediction and use Gaussian process to accelerate the search of the feature yielding the highest AUROC.

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist, CSAIL

Thesis Supervisor: Kalyan Veeramachaneni
Title: Research Scientist, CSAIL

Acknowledgments

This thesis would not have been possible without the guidance, encouragement and funding from my advisor, Una-May O'Reilly. I am especially grateful for the latitude at the beginning of my time in her research group to explore a wide range of topics and ideas which we gradually narrowed down to form this work. I am also extremely grateful for Kalyan Veeramachaneni's ideas and guidance throughout my time in the group and in the course of this project: as we worked very closely on several projects Kalyan naturally became my co-advisor.

This first half of this thesis includes text and experiments from the paper Dernoncourt et al. (2013c): Una-May and Kalyan played a key role in defining the problem to be addressed as well as pinpointing the most impactful approach. Alexander Waldin and Chidube Ezeozue helped to extract the data from MIMIC II, Max Kolysh wrote the script that validates the beats, Julian Gonzalez helped to extract features, Prashan Wanigasekara and Erik Hemberg gave me a hand to use the NFS, Bryce Kim and Will Drevo assisted me in using the OpenStack computer cluster, and Dennis Wilson gave some useful pointers to deal with the Matlab licenses.

The second half includes new experiments that will be published in a near future. Part of the results were presented at the MIT Big Data Initiative Annual Meeting 2014. In addition to his advisory role, Kalyan identified and defined the problem, and introduced me to a set of techniques including wavelets and Gaussian processes, which turned out to be essential and led to some very interesting results.

During these first two years at MIT, I have also worked on MOOC research under the supervision of Una-May and Kalyan. Our approach to MOOC shared many similarities with the work presented in this thesis. I had the pleasure to work with Colin Taylor, Sebastian Leon, Elaine Han, Zachary Pardos, Sherwin Wu, Chuong Do, Sherif Halawa, John O'Sullivan, Kristin Asmus, and many people at edX (Veeramachaneni et al., 2013; Dernoncourt et al., 2013a,b,d).

More generally, my research laboratory, EVO-DesignOpt (Evolutionary Design and

Optimization), now renamed as ALFA (Any Scale Learning For All) to reflect its focus on scaling up machine learning algorithms, was a fruitful environment and I feel lucky to have been surrounded by such great and diverse colleagues: Quentin Agren, Ignacio Arnaldo, Brian Bell, Owen Derby, Will Drevo, Chidube Ezeozue, Julian Gonzalez, Elaine Han, Erik Hemberg (with whom I had the pleasure to work on distributed evolutionary computation (Hemberg et al., 2013a,b)), Bryce Kim, Max Kolysh, Sebastian Leon, Zachary Pardos, Dylan Sherry, Colin Taylor, Alexander Waldin, Prashan Wanigasekara, Dennis Wilson, and Sherwin Wu. This intellectually fruitful research environment also helped me for my own side projects (Dernoncourt, 2012, 2014a,b).

Beyond my research laboratory, this thesis heavily relied on a 1,500-core OpenStack computer cluster and I thank the patience of MIT CSAIL technical members Jonathan Proulx and Stephen Jahl for answering to my dozens of bug reports and other miscellaneous issues, and the generosity of Quanta Computer, who donated a large part of the cluster hardware. Many thanks as well to Garrett Wollman for his Unix expertise and NFS server skills.

Outside MIT I was lucky to receive much advice from physicians David Dernoncourt and François De Forges, and some C++ strength from Paul Manners. In addition to one-to-one exchanges, I frequently used the Q&A communities Quora and Stack Exchange, which are two tremendous sources of information and great places to exchange ideas. I wish the research community followed such an open, collaborative, constructive model and I strongly hope that research will step-by-step adopt the principles of open science.

Most importantly, I thank my family and my wonderful girlfriend for their unconditional supports: moral, financial, technical and mathematical.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The focus of our work is methodological: we construct a general approach to make predictions from a raw data set of physiological waveforms, which we call BeatDB as it revolves around a database structure. We demonstrate this methodology by carrying out a set of experiments that take advantage of BeatDB applied to the problem of blood pressure prediction using the MIMIC II version 3 database. This chapter presents our objectives, the motivations behind our work as well as the main challenges we face.

1.1 Objectives

We want to simplify prediction studies on physiological signal datasets. A typical study, even with a modest number of patients, usually takes from 6 to 12 months. For that reason, physiological signal datasets have been by and large underexplored.

In response we design a large-scale machine learning and analytics framework, BeatDB, to scale and speed up mining knowledge from waveforms.

Our objectives for the framework are threefold:

- **Multi-level parameterization**: as many parameters as possible should be

changeable, even structural parameters such as lag and lead, so that any investigation should be feasible by simple parameterization.

- **Lossless storage**: the system should import existing data sets into its own format, which should conserve all the information needed for any future investigation. The underlying assumption behind this objective is any piece of information can turn out to be the key for a prediction problem.
- **Scalability**: physiological signal data sets can be massive, our framework should be able to scale to cope with any data set size.

We will demonstrate that our framework satisfies these objectives with a specific use case: predicting blood pressure, more specifically *Acute Hypotensive Episodes (AHE)*, with the MIMIC waveform database.

1.2 General prediction framework

With the above-mentioned objectives in mind, we design BeatDB's general prediction framework, which enables users to assess how much predictive power a set features contains with regard to an event. Events may be externally defined, i.e. via clinical data that are not physiological signals, or be detectable within the physiological signals. To accommodate inexact definitions of an event, the framework allows the user to parameterize an event. How features are aggregated and how the prediction is defined (lag and lead) can also be extensively parameterized. In detail:

Step 1: Define an event The user chooses an event (e.g. an acute hypotensive episode). BeatDB scans the data set to find signal records that contain the event and for each occurrence of the event identify the event's start and stop time indices. A part of the record that precedes the event's start time index becomes the signal that is used to make the event prediction and is called *lag*. The lapse of time between the end of the lag and the beginning of the event is called the *lead*.

Step 2: Define the data aggregation: The user chooses the length of the *lead* and *lag*. The lag can be divided into several windows. The user chooses which features to use, and the feature values are aggregated in a multi-level fashion that we will detail in later sections.

Step 3: Choose the machine learning algorithm: The user selects a machine learning algorithm, e.g. decision trees, SVM, logistic regression (the logistic regression is the only machine learning algorithm available at the time of the writing).

Step 4: Choose an evaluation metric for the prediction: The researcher selects an evaluation metric such as area under the ROC curve, Bayesian risk for a given cost matrix or Neyman Pearson criterion (the area under the ROC curve is the only evaluation metric available at the time of the writing).

BeatDB next sweeps the combined ranges of the parameters experimentally, or use a Gaussian process to orient the search to find the best parameter set. For each parameter set, it returns a result, which is the quality of the prediction according to the metric chosen by the user.

We will explain this framework in more details in the rest of the thesis and demonstrate it on a real use case, the prediction of *Acute Hypotensive Episodes (AHE)*.

1.3 General motivations

The root of our work lies in the tremendous size of medical data sets, or to put it in a much-hyped term *Big Data*. Beyond the commercial resonance of the term lies two core observations from a machine learning perspective:

1. More data usually increases the prediction accuracy.
2. As the data set size increases, the ranking of the prediction models can be shuffled. To put it otherwise, an algorithm can yield a higher accuracy compared with another algorithm on a data set of size x, while yielding a lower accuracy

on a data set of size $10x$.

Figure 1-1 illustrates those two aspects on the natural language processing task of confusion set disambiguation with a 1-billion-word training corpus.

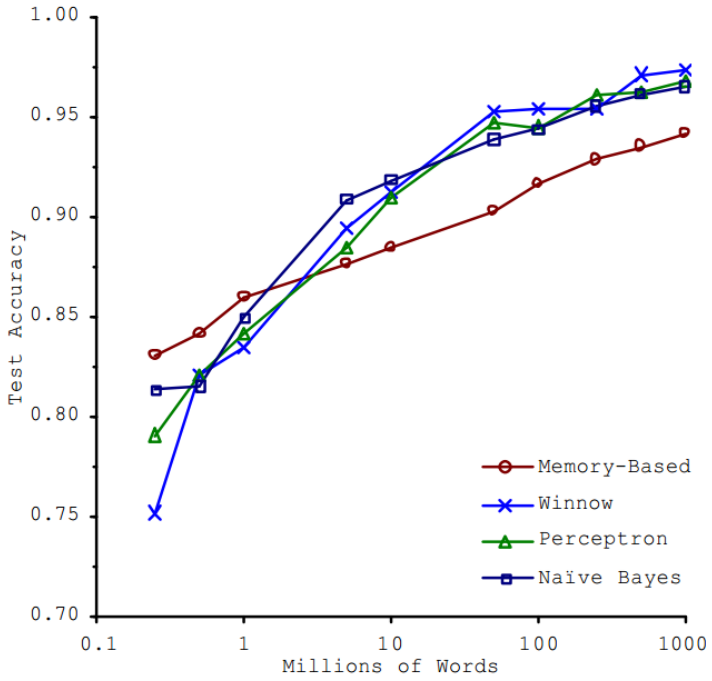

Figure 1-1: Learning curves of 4 different algorithms for the natural language processing task of confusion set disambiguation. A bad algorithm with more data can beat a good algorithm with less data, and the algorithm ranking is shuffled as the training data set grows. Source: Banko and Brill (2001).

We are now at a critical time in the history where the cost of storage has become cheap enough (see Figure 1-2) to easily store most data sets. Concomitantly sensors are becoming omnipresent in our daily life, as epitomized by the Quantified Self movement, which promotes the data acquisition of anything surrounding an individual, from food consumed to physiological data such as blood pressure, as explained in Swan (2013).

As a result, the amount of self-quantifying devices has skyrocketed over the last few years, either as wearable devices (Mann, 1997) or software applications: Fitbit Tracker, Jawbone UP, BodyMedia FIT, Samsung Gear Fit, Nike+ FuelBand, Pebble,

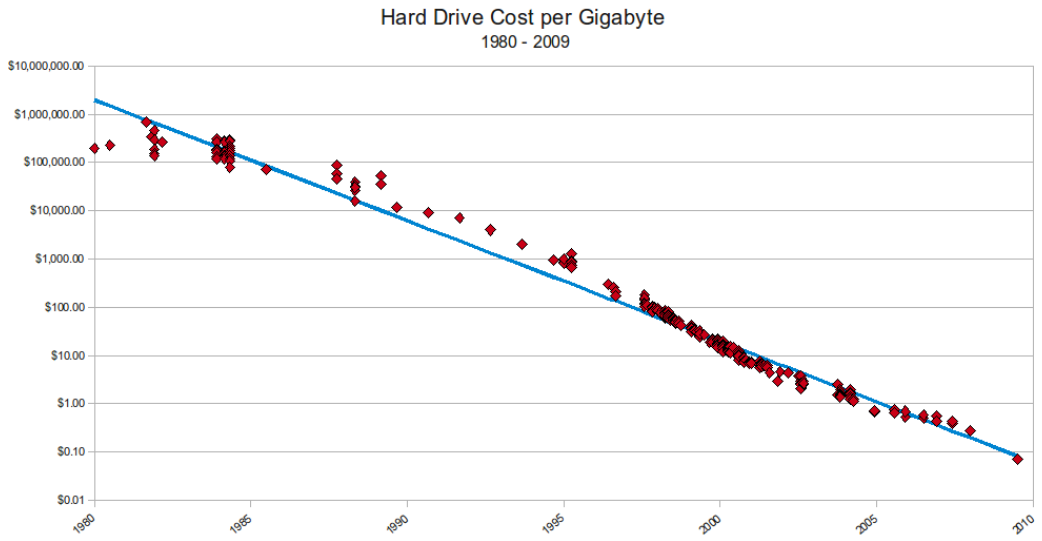

Figure 1-2: Evolution of storage cost: 1980-2010. Source: Komorowski (2009).

Technogym, WakeMate, Zeo, MyFitnessPal, etc. People's willingness to share huge amounts of their data represents a tremendous opportunity for scientists, and in particular from an applied machine learning perspective. Tung et al. (2011) report that amongst the customers of personal genomics and biotechnology company 23andMe, close to 90% consent to participate in research, and around 80% choose to contribute additional phenotypic data by answering research questions.

However, the main issue in big data often doesn't stem from the sheer size of data but from its noisiness as well as its poorly structured or even raw nature. As a result, many data sets remained un- or under-explored. From a machine learning standpoint, such unstructuredness causes researchers to create their own ad hoc, temporary structure designed for a specific experiment. This might hinder reproducibility and comparability as each researcher is working with their own tools and pre-processed data, and it certainly hinders the amount of experiments that can be done within a given time window in comparison with a situation of shared development efforts to create to common, flexible framework.

Appendix B discusses privacy and anonymization of personal data.

1.4 Technical challenges

The main technical challenge of this work is to handle the amount of data as well as its raw nature. The latter requires extensive pre-processing that we will detail later on. The former forces us to use a distributed file system, since one single hard drive is not enough to contain the data set. It also leads to a high computational cost even for simple operations on the data, which require us to use a computer cluster to be able to perform them in a reasonable amount of time. The technical details will be exposed in the following chapter.

Each of our three objectives that we have presented in Section 1.1 has its technically challenging counterpart:

- The multi-level parameterization means that the system should be designed in such a way that most parameters can be changed, which implies a highly flexible system design.
- The lossless storage inevitably leads to large data size.
- The scalability objective can only be reached by implementing a distributed, multithreaded solution.

Beyond the data set itself and the technical challenges, our work is at the intersection of machine learning, digital signal processing, databases and medicine: unifying those different fields into a common project was a challenge on its own, as each comes with a particular culture, set of competences as well as different software and theoretical approaches.

1.5 Contributions

The contributions of this thesis are threefold:

- *Data set size*: as far as we know this is the first time that all patients from the

MIMIC Waveform data set were used to perform event prediction.

- *Scalable, flexible system*: every level of our system is designed to be scalable when deployed on an OpenStack computer cluster, and has its own a set of parameters that can be easily modified.
- *Meta-heuristic layer*: our system contains a meta-heuristic layer for feature discovery.

Our contributions are both methodological and experimental as we demonstrate our system on a real-world use case.

1.6 Organization

The rest of this thesis is organized as follows:

- Chapter 2 presents the BeatDB framework we created with a focus on the architecture design and a few technical aspects.
- Chapter 3 presents the MIMIC data set, which is the data set that we use to demonstrate BeatDB.
- Chapter 4 demonstrates BeatDB with a prediction problem, namely predicting the acute hypertensive episodes of patients in intensive care unit.
- Chapter 5 demonstrates BeatDB with the same prediction problem as in the previous chapter but using wavelets.
- Chapter 6 shows how BeatDB uses a Gaussian process for parameter optimization for the same prediction problem.

Chapter 2

BeatDB

In this chapter we present BeatDB: its organization as well as its system design. We will demonstrate with a real-world use case in the subsequent chapters.

2.1 Definitions

A *signal* is a series of *samples*. The terms *sample* and *sample values* can be used interchangeably. In a physiological signal dataset, signals are typically grouped by *records*. A record contains all the samples recorded by a *sensor* for a specific signal on a continuous period of time. Since sensors can be unreliable, a record might contain *jumps*, i.e. periods of time during which no sample is recorded, and samples are typically *noisy*, i.e. the sensor add some noise to the ground truth sample.

A *physiological signal* is the signal recorded by a *sensor* placed on the body of a living being or implanted (Hamid Sheikhzadeh, 2007).

A physiological signal data set is a set of records, usually organized by:

- signal type, such as blood pressure,
- location where the signal was measured, such as John Doe's radial artery.

2.2 Schema

The key unit of many physiological signals is the *beat* (or pulse), which is a much more meaningful unit to physicians than samples. Furthermore beats are the fundamental periods of the signal. As a result, BeatDB is organized around beats and features are computed at the beat-level. Beats offer a fine-grained perspective of the signal: even though it means that the resulting data take a significantly large amount of storage space, we do not want to store some aggregation of several beats instead, such as storing the average of some beat feature over 1-minute period, as we could lose some precious information. Amongst the objectives we set for the platform is lossless storage: any piece of information can turn out to be critical for a prediction problem, and storing aggregations instead of each individual beat would make the platform useless for some prediction problems.

As a result, the first step to make the signals more exploitable is to detect the beat onsets. Furthermore, the recording of a signal sometimes contains jumps, that is to say that the signal was not recorded for a certain lapse of time. We can detect such jumps as each sample has a timestamp. We therefore add a third type of beat in addition to valid and invalid beats: jumps.

Lastly, once every beat has been detected and marked as valid, invalid or jump, we compute a series of features that aim at characterizing them. A feature is a function that takes as input the samples of one beat, and outputs one real number which we hope might contain some useful information about the beat, useful meaning that it may help the machine learning techniques to predict a certain event. Features are designed to extract some salient information about the beat. We will detail in Section 2.3 which features we compute for the use case we demonstrate.

As a result, for each record we store:

- the list of all samples. At first we used to store them in one CSV file with column 1: sample ID; column 2: sample value. But this turned out to be inefficient in terms of storage space so we changed the organization of the file to a new

format where each row corresponds to one beat and contains all sample values of the beat. In the use case that we will demonstrate later, this simple trick allowed us to reduce the size of sample value files from 500 GB to 100 GB, hence reducing the storage cost, network bandwidth and CPU cost, as those files are processed by a computer cluster and compressed.

- the list of all beats, which we characterize with three properties and store in another CSV file. Column 1: sample ID of the first sample in the beat; column 2: sample ID of the last sample in the beat column 2; column 3: beat validity (valid/invalid/jump).
- for each feature we store the feature value of every beat in one file. Hence if we compute 10 features we have 10 different files.

Tables 2.1 and 2.3 show a short example of each of those three files. Figure 2-1 shows an overview of the feature database of BeatDB.

Appendix C explains our choice to use flat files instead of using a relational database management system (RDBMS), and in particular the advantages of storing data in a column-oriented fashion instead of the more traditional row-oriented organization.

Sample ID	Sample value
15684	108.8
15685	105.6
15686	103.2

Table 2.1: First version of the structure of the record raw sample file. Each line contains a single sample. Every record has its own file, which is why there is no record ID column.

Sample values: each line contains one entire beat)										
68.1	72.8	78.4	85.6	93.6	98.1	95.3	91.2	91.1	...	variable length
74.4	79.2	85.6	92.8	100.8	108.8	115.2	121.6		...	
73.6	77.6	84.0	91.2	100.0	108.8	117.6	124.8		...	

Table 2.2: Second version of the structure of the record raw sample file. Each line contains all the samples of one beat. As in the first version of the structure of the file, every record has its own file, which is why there is no record ID column. The new version allows to save 80% of disk space, and make feature computation easier as each feature is computed over one entire beat's samples.

Sample ID start	Sample ID end	Flag
46762	46863	0
46864	46961	1
46962	47064	1

Table 2.3: Record validation file. The file aims at flagging which beat are valid or invalid. Each line corresponds to one beat. The first two columns contain the temporal location of the beat, and the third column indicates the beat validity: 0 means the beat is invalid, 1 means the beat is valid, and 2 means there is a jump in the record's time series. As in the record raw sample files, every record has its own validation file, hence the absence of a record ID column.

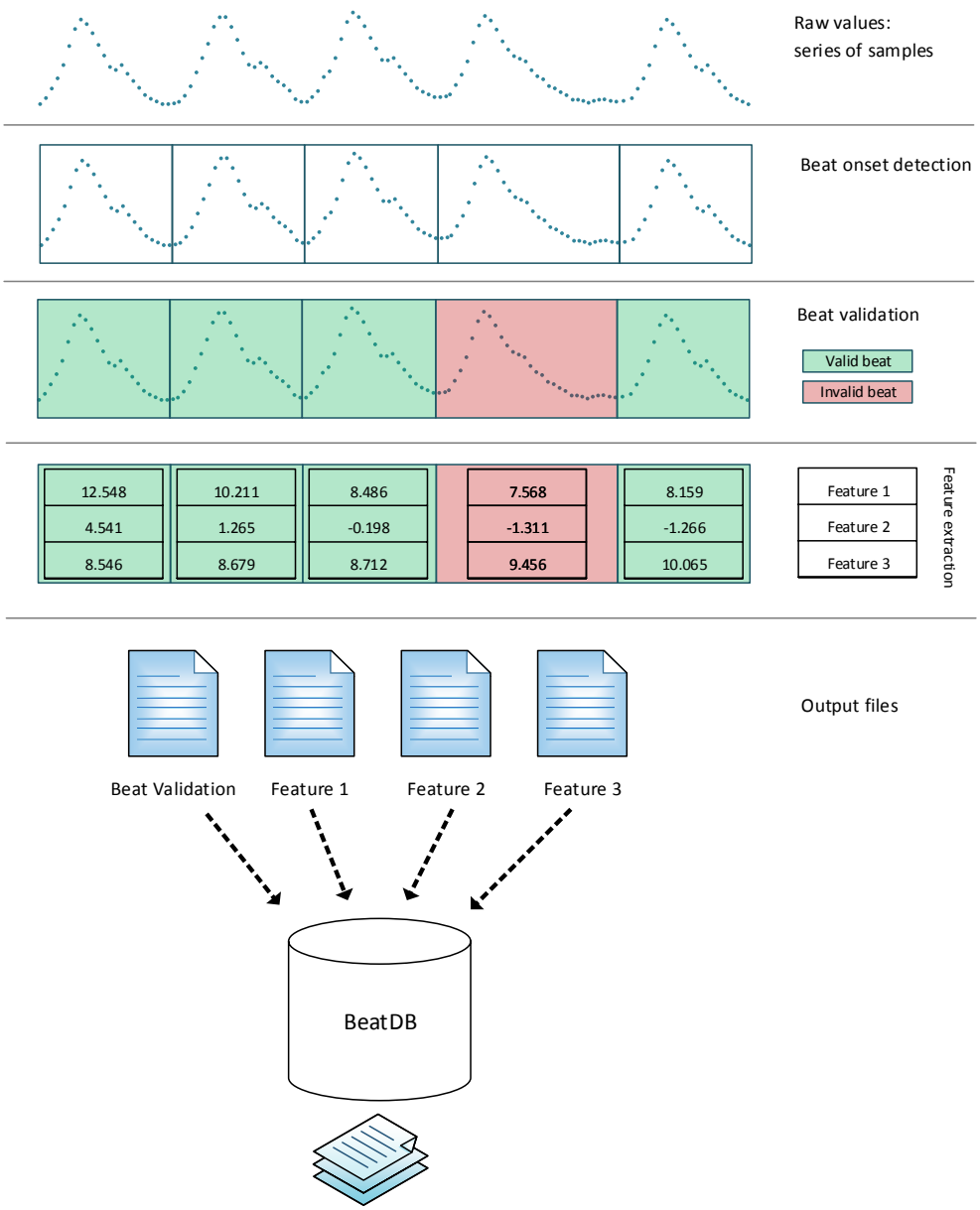

Figure 2-1: BeatDB overview: detecting the beat onsets, validating each beat and extracting features. Add data is stored is flat files, as described in Section 2.2.

2.3 Condition scanner

A condition may be externally defined, i.e. via clinical data, or be detectable within the signal data. In case the condition is defined based on the signal data, BeatDB has a scanner component in which we can define medical conditions and vary their parameters. The scanner will return all the periods of time where the condition occurs as a CSV file where the first column corresponds to record ID, the second column is the beat sample ID where the condition starts and the third column corresponds to the sample ID where the condition ends. Table 2.4 shows an excerpt of this CSV.

Record ID	Sample ID start	Sample ID ends
3001937	9341485	9592802
3001937	9387199	9691790
3001937	9470969	9707036
3003650	3562407	4068228

Table 2.4: BeatDB condition scanner output. Each line corresponds to one AHE event. The first column indicates the record under consideration, and the next two columns specify at what time the event occurred.

2.4 Prediction parameters

Beyond the parameters of the medical condition, the prediction problem has its own parameters:

1. The lag is expressed in time units (e.g. in minutes) and corresponds to the amount of data history we allow the model to use when making the prediction.

2. The lead is expressed in time units and corresponds to the period of time between the last data point the model can use to predict and the first data point the model actually predicts.

3. The prediction window is expressed in time units and corresponds to the time window we consider when looking for the occurrence of a medical event.

Figure 2-2 illustrates those three parameters that are specific to the prediction

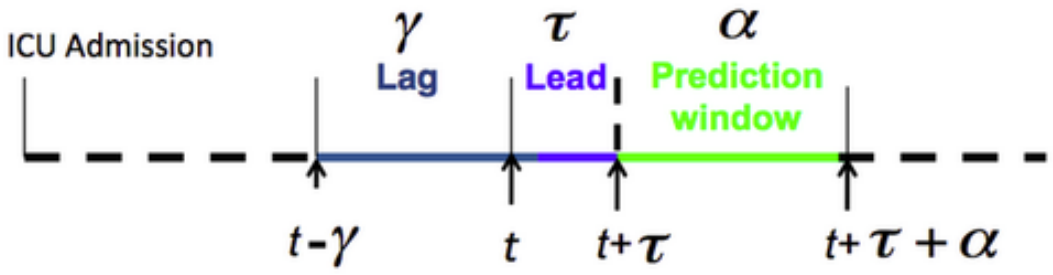

Figure 2-2: Prediction parameters. t represents the current time. The prediction window corresponds to the duration of the event we try to predict. The lead indicates how much time we try to predict ahead. The lag is the period of time in the past that we use from to compute the prediction.

2.5 Data assembling

Once we have both the conditions and the selected features, we can compile the data set, which will in turn be given to the machine learning algorithm that we will train to predict the condition (which is AHE in our case). The data set is compiled using the algorithm detailed in Algorithm 1. Figure 2-3 represents the algorithm visually. Figure 2-4 shows the data assembly algorithm with aggregation functions, which can be optionally defined on top of sub-aggregation functions.

As we can see, compiling the data set has its own set of parameters:

- which feature(s) to use,
- how many sub-windows the lag should contain,
- which aggregation function(s) to use,
- how large should the sliding window be.

Table 2.5 summarizes all the parameters handled by BeatDB.

Input: Data files, lead, lag, window_slide, number_of_subwindows_in_lag
Output: Compiled data set contained one CSV file

```
1
2  condition_file = open('condition_file.csv', 'r') // Open necessary files
3  output_file = open('output_file.csv', 'w')
4
5  for each record do
6      open the feature files, the duration file and the validation file
7      for each occurrence of the condition for the record do
8          in each feature file, set cursor_position to the sample ID start of the
           occurrence of the condition.
9          extract_row(cursor_position, True)
10     end
11     while True do
12         set cursor_position on first beat in the record
13         if far away from condition occurrences then
14             extract_row(cursor_position, False)
15         end
16         cursor_position = cursor_position + window_slide
17         if cursor_position is after the end of the file then
18             break
19         end
20     end
21 end
22
23 function extract_row(cursor_position, is_condition_present)
24     row = list()
25     compute lag_first_beat and lag_end_beat
26     for each feature do
27         extract the feature values between lag_first_beat and lag_end_beat
28         split this list of feature values into number_of_subwindows_in_lag lists
29         for each sublist l do
30             for each aggregation function f do
31                 row.append(f(l))
32             end
33         end
34         row.append(is_condition_present)
35         output_file.write(row)
36     end
37 end
```

Algorithm 1: Feature aggregation algorithm

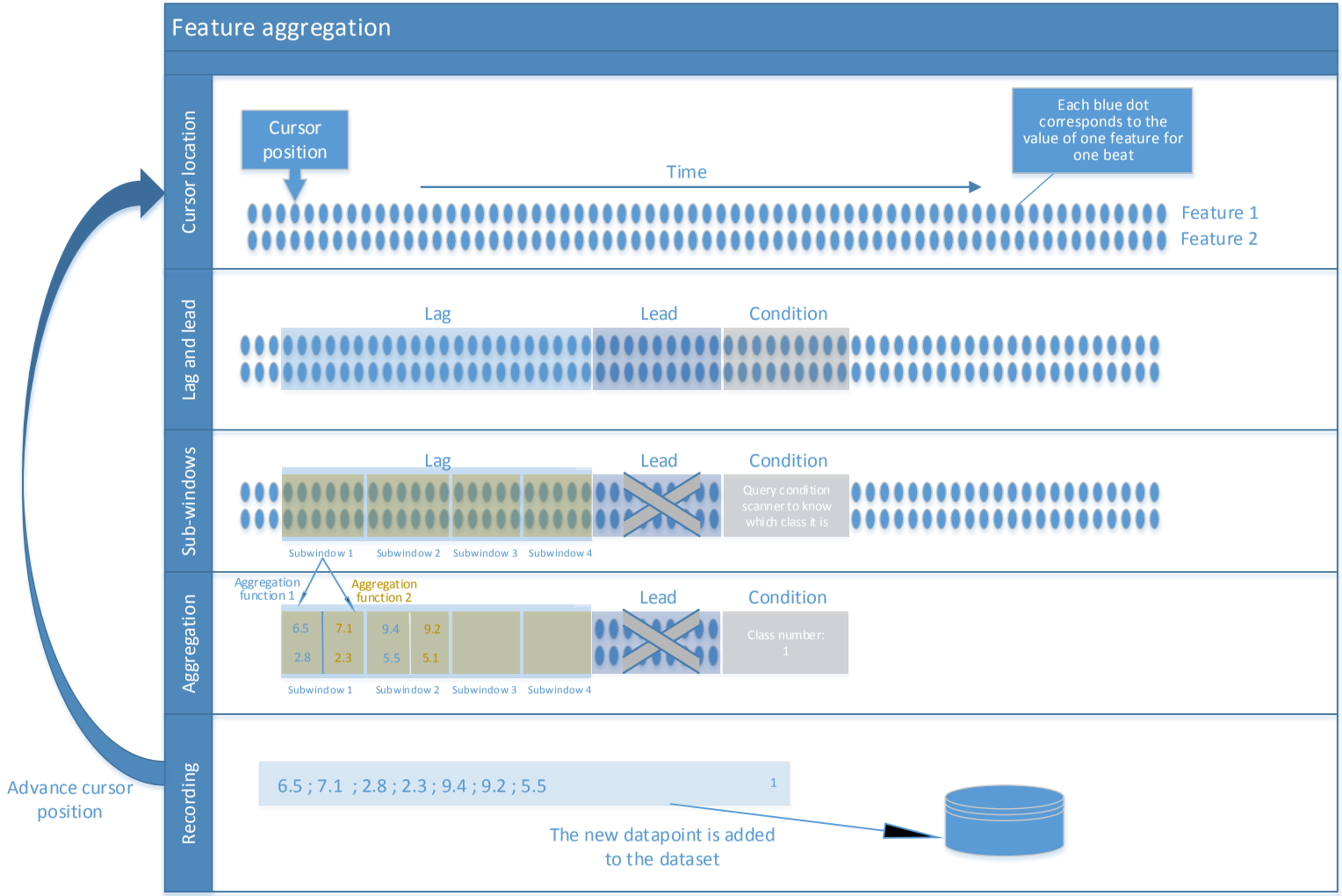

Figure 2-3: Visual representation of the feature aggregation algorithm

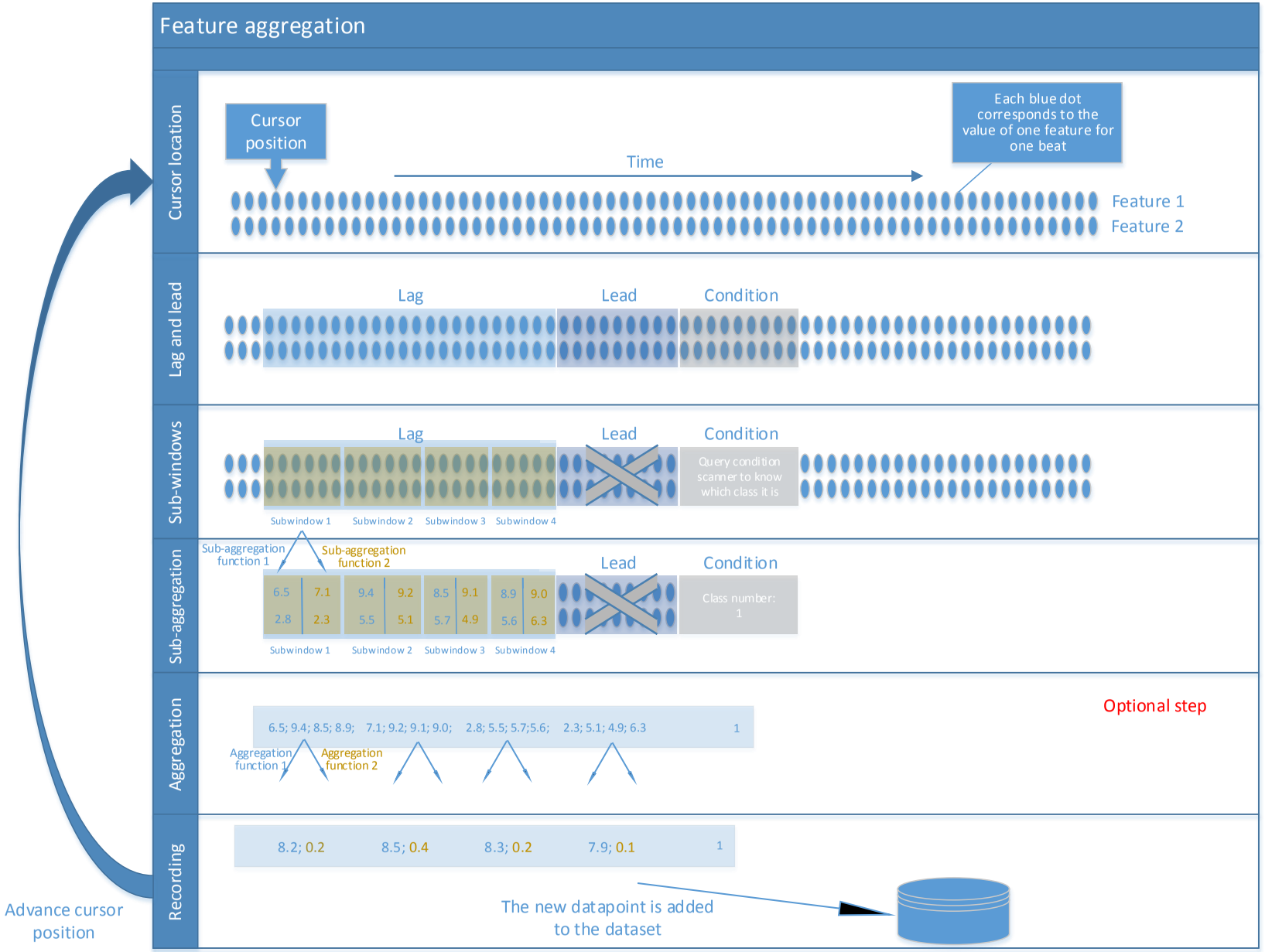

Figure 2-4: Visual representation of the feature aggregation algorithm. Aggregation functions can be optionally defined. Aggregation functions can be the mean, kurtosis, etc., just like for sub-aggregation functions. When aggregation functions are used, the final values are aggregated over the entire lag, and not just over sub-windows.

Parameter categories	Parameter names
Condition definition	Window size, threshold, frequency, variable
Prediction	Lag, lead, features
Data aggregation	Sub-aggregation window, sub-aggregation function, aggregation functions

Table 2.5: General prediction framework's parameters. The parameters can be divided into three main categories: the parameters that are specific to the condition's definition, the parameters that belong to the prediction problem's statement, and the parameters that are used during the aggregation of the data.

2.6 Event prediction

Once the data is assembled, BeatDB can learn a model using logistic regression, and assess its quality by computing the AUC of the ROC (area under the receiver operating characteristic, aka. AUROC) using cross-validation. Appendix D.1 explains how logistic regression works, and Appendix D.2 presents the AUROC.

2.7 Parameter selection

BeatDB can explore the parameter space using to different type of search:

- Grid search: the parameter space is explored exhaustively.
- Gaussian process regression: the parameter space is explored using a Gaussian process regression. See Appendix E for an explanation of the theory, and Algorithm 2 for the actual algorithm algorithm we use.

2.8 OpenStack and NFS

Given the size of the data as well to the amount of experiments that are carried out for this thesis, we use the MIT CSAIL OpenStack computer cluster as well as a 5TB NFS storage.

OpenStack is a free and open-source software cloud computing platform. A history of the OpenStack project can be found in Slipetskyy (2011). The MIT CSAIL OpenStack cluster contains a total of 768 physical cores using Intel Xeon L5640 2.27GHz chips (ca. 6,000 virtual cores), 10 Dell r420 servers with dual socket 8 core E5-2450L (ca. 1,200 virtual cores), and 5 TB of RAM. In the following chapters, we will specify for each experiment how much resource we use.

As OpenStack workers can mount NFS filesytems and our network has a 10 Gbit/s link between NFS servers and OpenStack servers, we make a heavy use of this connection in order to perform computation using OpenStack workers' CPU and RAM on data retrieved from the NFS filesytem (BeatDB data).

2.9 Distributed system architecture

We use two different models of communication to distribute computation over the OpenStack cluster:

- Master/worker pattern: we use dcap presented in Waldin (2013a) which provides a framework to distribute tasks among workers, each worker being an OpenStack instance. The master is another OpenStack instance that contains the list of tasks to assign. The server listens to any request from workers asking for a new task, and gather data when the task is done. Figure 2-5 presents the master/worker architecture of dcap.

- Multi-worker synchronized via a result database: even though the master/worker model is fairly simple, it does require a non-negligible coding overhead, essen-

tially to handle connections (listening, data transfer, being robust to connection issues, etc.). We therefore changed the architecture over the course of the project to a multi-worker architecture, where each OpenStack instance retrieve a task by querying a database. To ensure that two instances do not retrieve the same task, an instance writes a flag in the database to indicate that it is working on the task. When the task is done, the instance writes the result in the database, and, if needed, writes files on the NFS filesystem. Figure 2-6 represents this multi-worker design synchronized via database.

In the rest of the thesis, we will only use the multi-worker system architecture. In this setting, putting aside the database server, each machine can be interchangeably called worker, node or instance. We will use the term worker.

The result database contains all the results returned by the workers. By fetching the result database content, the workers make sure not to compute a parameter set that has already been done, since whenever a worker starts a task it adds a flag in the result database. If the workers use a Gaussian process regression to decide which parameter set to compute, they use the result database content to fit the Gaussian process.

Figure 2-6 presents a visual representation of the algorithm we use when several machines are used to compute a grid search or a Gaussian process. In the example presented in the figure, we suppose that there are two workers computing tasks through the grid search or the Gaussian process, Worker #1 and Worker #2. Worker #2 is computing task #3, while Worker #1 is not computing any task, either because it was just launched or the previous task was completed. We go through the example presented in the figure:

1. Worker #1 is looking for the next task to compute. For that purpose, it needs to retrieve all the task results that had been previously computed from the database server.

2. Once Worker #1 has retrieved the task results, it either selects the next param-

eter set to be done according to the grid search, or it fits a Gaussian process in order to the determine what is the most promising parameter set to compute next. Worker #1 makes sure that no other worker is currently computing this exact same parameter set by checking in the database whether parameter set has been flagged as being computed (the flag is a -1 in the result_value column). If another worker is computing the same parameter set, then Worker #1 selects the second most promising parameter set. If it is also taken, then it selects the third one and so on until it finds a parameter set that is not being computed. If Worker #1 cannot find such parameter set, then it means that the search is over (all parameter sets have been or are being computed) and Worker #1 terminates.

3. Worker #1 leaves a flag in the database so that no other worker can compute the same parameter set: if for example a third worker is launched right after and finds its next task to compute by fitting its Gaussian process using the existing task results, it will choose the same parameter set as Worker #1 because our algorithm to find the next new task is deterministic. In this case, the third worker will have to choose its second best choice of parameters (or third, fourth, etc. depending on which parameter set remained be done, as we have seen in the previous step).

4. Worker #1 computes its task, task #5.

5. Upon completion of its task, Worker #1 writes the result in the database.

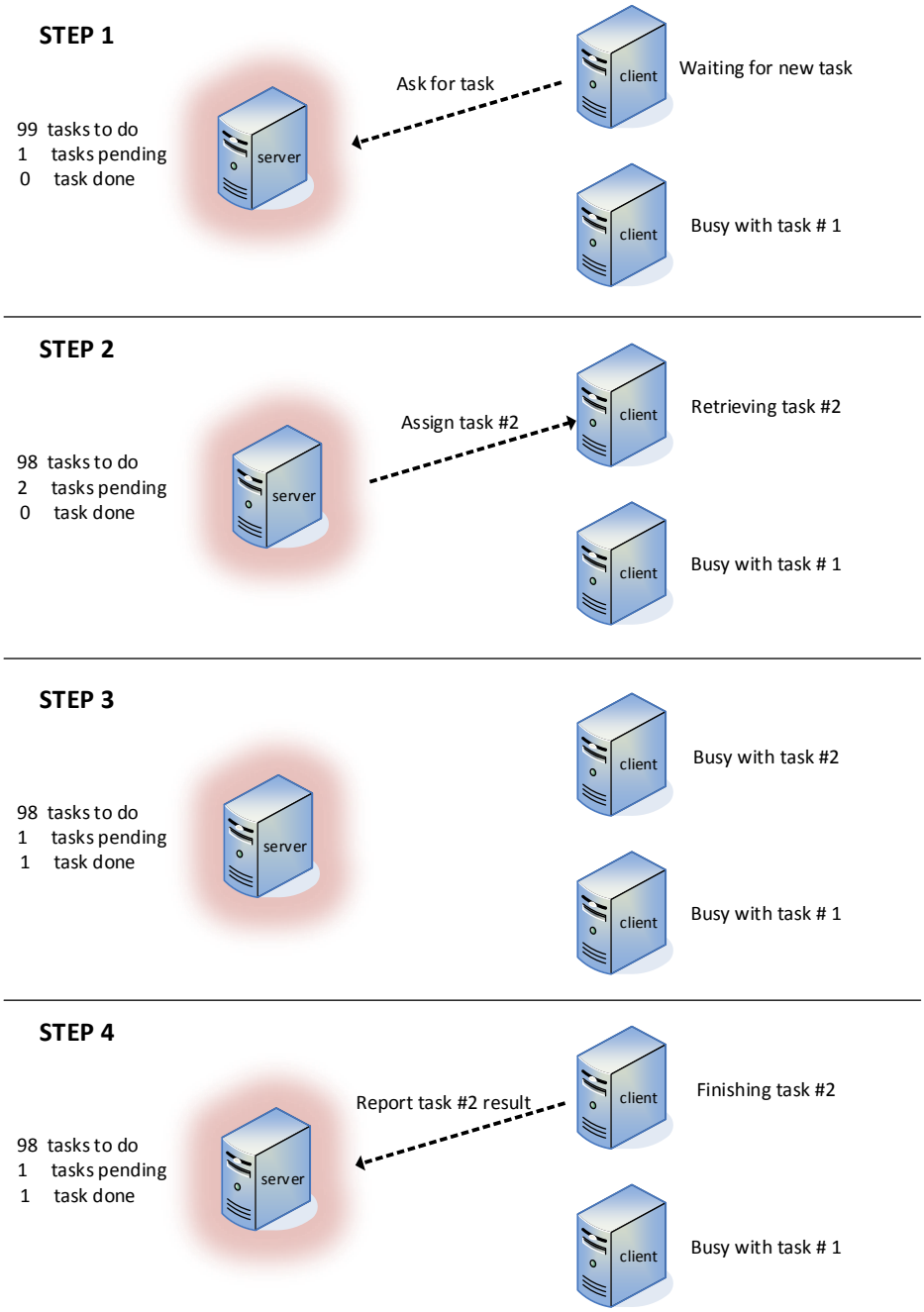

Figure 2-5: Master/worker architecture: dcap. The master contains the list of all tasks that need to be done. Whenever a worker is idle, it asks the master for a new task. The master sends the task to the client, the latter performs the task, at the end of which it return the result to the master, and becomes idle again. The master must continuously listen to a port so that workers can ask for new tasks and return results.

Figure 2-6: Multi-worker architecture synchronized via a result database: grid search or distributed Gaussian Process.

2.10 Worker logic

In earlier sections we explained the parameter selection (Section 2.7), data assembly (Section 2.5) and event prediction (Section 2.6). We also presented the distributed system architecture (Section 2.9) and show how workers interact with the rest of the system. In this section, we explain the logic of each worker.

Each worker perform the same three-step cycle: parameter selection, data assembly and event prediction. At the end of each cycle the prediction's quality, namely the AUROC, is saved in the result database. The time it takes to perform such a cycle is called the *worker cycle time*. A worker stops when the parameter space has been exhausted or when it gets killed by the cleaning script that we will explain in Section 2.11.

Figure 2-7 shows the worker logic when parameters are selected using a Gaussian process regression.

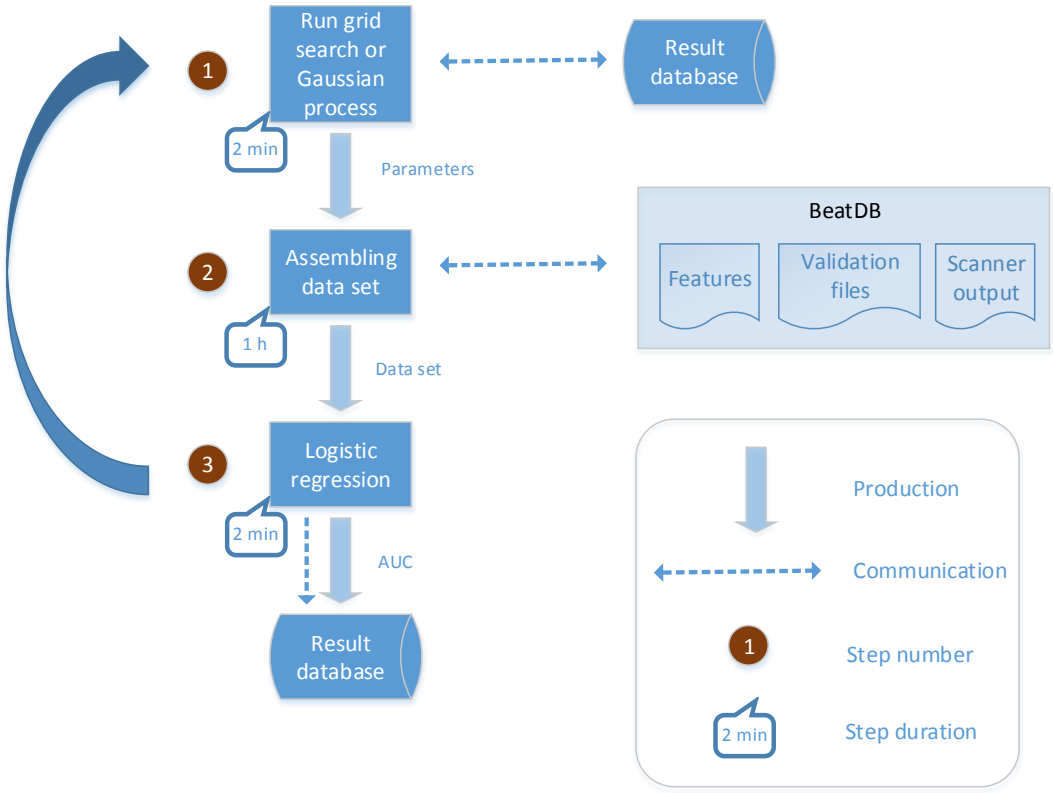

Figure 2-7: Worker logic. Each worker continuously performs a three-step cycle. The step is to decide which parameter set the next task to compute will have. This step involves fetching all the current results from the result database, which the decision of the parameter set will be based on. The second step is to assemble the data according to the chosen parameters. This step requires the worker to retrieve data from BeatDB: to assemble the data the worker needs to retrieve feature files, validation files and condition scanner outputs. Figure 2-4 details how the data assembly is done. Lastly, the third step for the worker is to perform predictions by fitting a regression curve using logistic regression. The prediction quality we chose is the AUROC. This AUROC along with the problem parameters are saved in the result database. Once the AUROC result is saved, the worker has completed its cycle and goes back to the first step. The duration times displayed on this figure are for a specific experiment, which we will detail later. The message conveyed by these duration times is that steps 1 and 2 are fast while step 2 is much longer.

2.11 Cleaning broken workers

We dedicate up to 150 4-core workers for our experiments, the exact number of instances at a given time depending on the resource availability, and experiments can take up to several days. Running a large number of instances for a long period is interesting from a system standpoint as a large variety of issues can appear. Here are a few ones that we experienced throughout the project:

- Router issues
- OpenStack DHCP server issues
- Hard drive failure
- Connection issues with the NFS server
- Connection issues with the Matlab license server
- Instances being shutting down due to the hypervisor running out of memory and running the oom_killer process (out-of-memory killer) without any warning.
- High CPU Steal Time due to OpenStack being configured to overcommit CPUs by a 4:1 ratio (i.e. there are 4 times more VCPUs than physical ones).
- Thrashing on some hypervisors on Openstack due to OpenStack being configured to overcommit the RAM by a 1.5:1 ratio and because of intensive I/O work.

Remarkably we avoided network congestion issues thanks to the size of the network links (10 Gbps).

In order to face this multi-level instability, we run on a 1-core instance a daemon process that continuously checks each instance and make sure it has returned a result over the past X hours, X being determined depending on the cycle time a worker takes on a healthy instance. If not, the instance is terminated and a new one is launched. In order to avoid the High CPU Steal Time and Thrashing issues, we sometime reduce the X hours threshold so as to terminate slow instances, in the hope

that newly launched instances are allocated to some less stressed hardware resources, as shown in Figure 2-8.

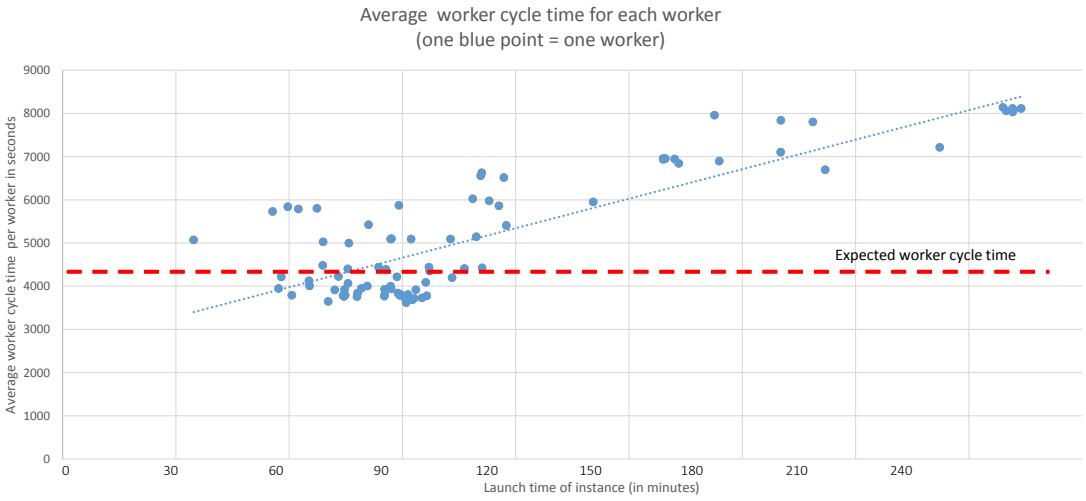

Figure 2-8: Average worker cycle time. Each blue point we present one instance. The worker cycle time on a 100% healthy, unstressed instance should take less than 4000 seconds. The plot shows that the later the instance is launched, the more stressed the hardware resources on which the instance was launched are.

2.12 Conclusion

In this chapter we have presented BeatDB, which we designed to be both flexible and scalable. The flexibility stems from its multi-level parameterization: condition definition, data assembly, feature selection and search method over the parameter space. The scalability aspect was considered from the beginning and the algorithms are distributed.

We will demonstrate BeatDB in the next chapters with a real use case: blood pressure prediction using one of the largest physiological signal data set publicly available, MIMIC.

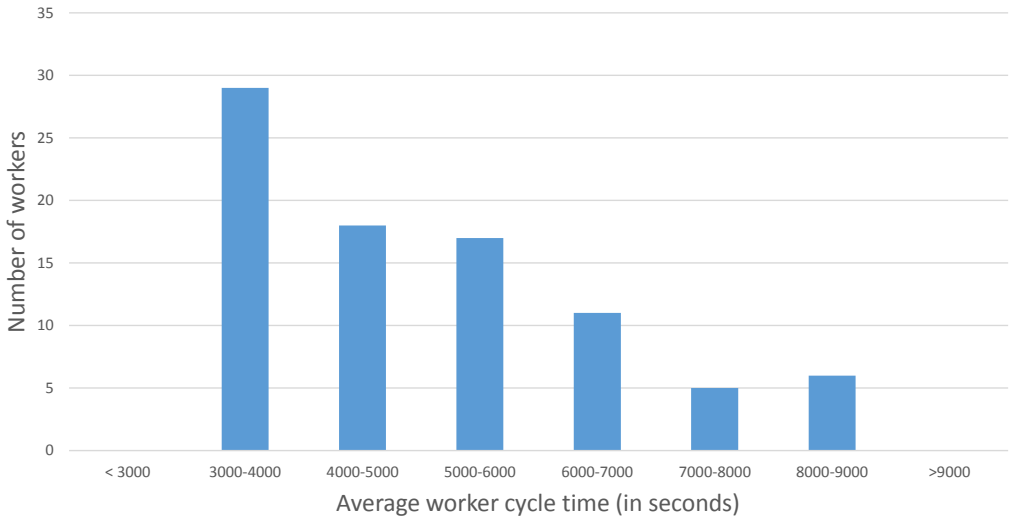

Figure 2-9: Histogram of the average worker cycle time per instance over time.

Chapter 3

The MIMIC data set

This chapter presents the data set that we will use in the following chapters for a real use case of BeatDB, MIMIC. It also presents the signal on which we will focus for both the features and the medical condition that we will predict: the arterial blood pressure (ABP).

3.1 MIMIC

We use the *Multiparameter Intelligent Monitoring in Intensive Care II* database (MIMIC II) version 3, which is available online for free and was introduced by Moody and Mark (1996) and Goldberger et al. (2000). In order to protect patients' privacy, data was de-identified[1] using customized software developed for that purpose (Neamatullah et al., 2008): dates were shuffled, names removed and a few other techniques were used. Figure 3-1 presents an overview of the database.

MIMIC II is divided into two different data sets:

- the *Clinical Database*: it is a relational database that contains many information about ICU patients such as patient demographics, hospital admissions and

[1]De-identification differs from anonymization in that the latter is supposed to be irreversible while the former may be re-identified by a trusted party.

discharge dates, room tracking, death dates, medications, lab tests, notes by the medical personnel, and so on. A script is provided to pipe the data into a PostgreSQL database.

- the *Waveform Database*: it is a set of flat files that contains 22 different kinds of signals for each patient. Each flat file is stored using the PhysioBank-compatible (aka. WFDB-compatible) format, which is a format specific to the organization that manage MIMIC PhysioNet (2014). We will focus on this data set only in the rest of the thesis.

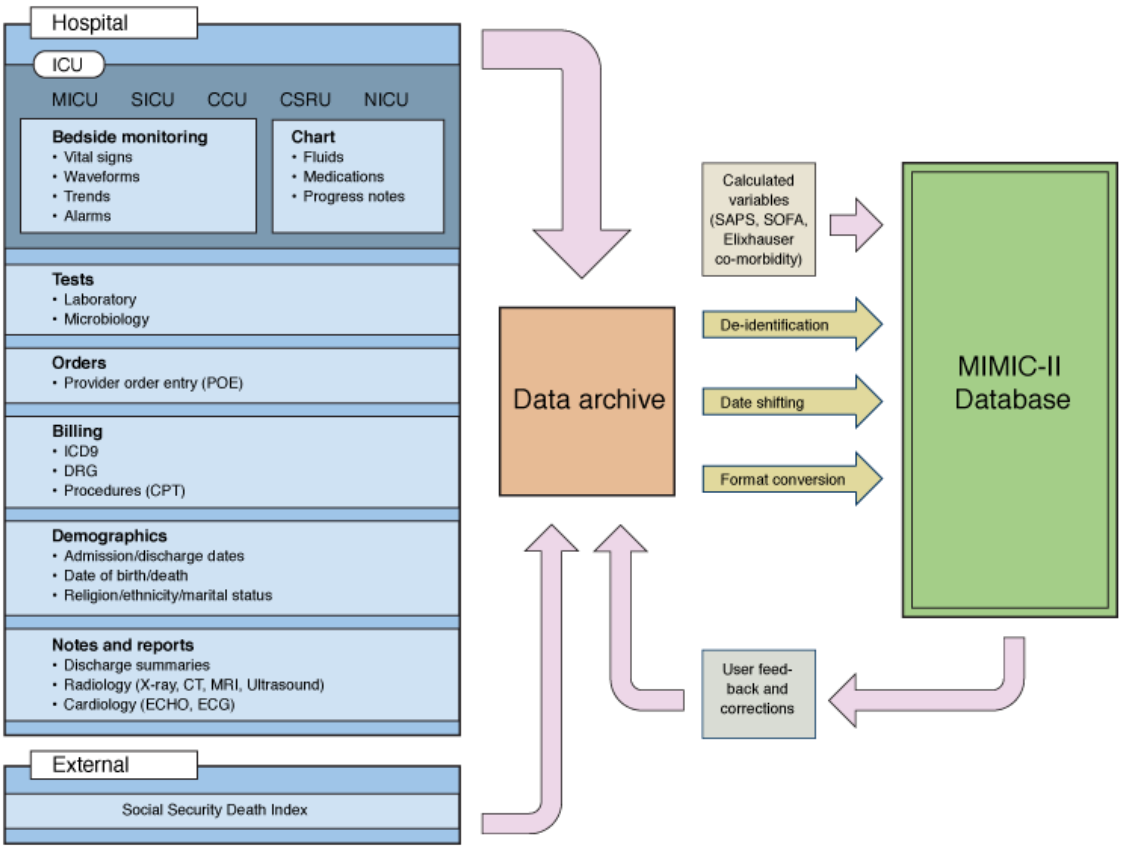

Figure 3-1: MIMIC-II Database organization

The Waveform Database gathers 23,180 sets of recordings and over 3 TB of data. One recording is typically one patient, but in some rare occurrences one recording can contain several patients when the medical personnel forgot to change the patient ID, or one patient might be split in several recordings in case the patient came to the

ICU several times.

Among the signals, some were recorded at 125 samples per second, such as ECG (electrocardiographic) and ABP (Arterial Blood Pressure), other signals were recorded at 1 sample per second, such as the cardiac output, heart rate and the respiration rate. We chose to use the ABP for this study, but our framework can be extended to any other signal.

There are 6,232 patient records that contain ABP. To have a sense of the massiveness of the data, since the signal was recorded at 125 Hz and we had a total of 240,000 hours of ABP data, we have 108 billion samples ($240000 \times 60 \times 60 \times 125$). ABP samples are measured in mmHg (millimetres of mercury). Figure 3-2 shows 5 ABP beats along with their properties.

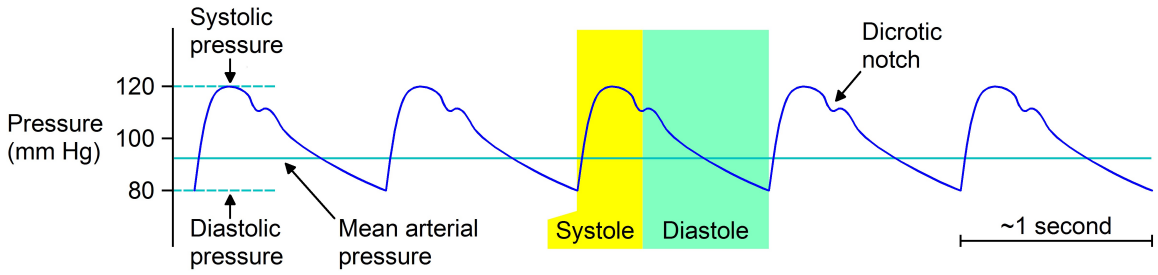

Figure 3-2: Arterial blood pressure (ABP) fluctuations. This figure shows 5 ABP beats and some usual properties. The ABP is measured in millimeter of mercury (mmHG). The systolic pressure corresponds to the maximum level of mmHG reached in a beat, while the diastolic pressure is defined as the minimum level of mmHG reached in a beat. A person's ABP is usually expressed in terms of the systolic pressure over diastolic pressure: for example, in the first beat, the patient has 120/80, which is a typical blood pressure. The systole is the period of time when the heart contracts itself to send the blood to the rest of the body (the term systole etymologically mean contract). This explains why the pressure increases during the systole. The diastole is the period of time when the heart refills with blood. Between the systole and the diastole, there is a brief interruption of smooth flow due to the short backflow of blood caused by the relaxation of the ventricle: this event is called dicrotic notch. Source of the figure: PhysiologyWeb (2011).

3.2 Arterial blood pressure measurement

The ABP is measured in an invasive way from one of the radial arteries of the patient, namely using arterial catheterization, as illustrated in Figure 3-3. In order to measure the blood pressure, the physician first inserts an intra-arterial catheter (aka. arterial line, or in short A-line) into an artery of the patient[2] , such as the radial artery (most common, as in MIMIC ABP), the brachial artery, the femoral artery, the dorsalis pedis artery or the ulnar artery. Figure 3-4 shows that the measurement location has a direct impact on the blood pressure values. The A-line is connected to a tube filled with a saline solution, which is connected to a pressure bag. A pressure transducer (aka. pressure sensor) is placed in the tube, and converts pressure into an analog electrical signal.

As detailed in Gomersall (2014), McGhee and Bridges (2002) and Nickson (2014), the process of measuring the ABP contains several sources of potential errors:

- Transducer: the position of the transducer influences the measured values: whenever patient position changes, the transducer height should be accordingly modified. Also, the transducer must be accurately leveled to the atmospheric pressure.
- Clotting in the arterial catheter: blood clots might form on the tips of arterial catheters.
- Tubing: there must be no air bubble in the tubing.
- Damping degree: all hemodynamic monitoring systems are damped, which means that the amplitude of the signal has been reduced. As shown in Figure 3-5, the damping degree must be carefully chosen.
- Device failure: a component might start malfunctioning for some technical reason, independently of the actions of the nurse or the physician.

[2] Srejic and Wenker (2003) present a series of pictures that show how to place an A-line. The insertion is most frequently painful for the patient: some anesthetic is often prescribed to reduce the pain.

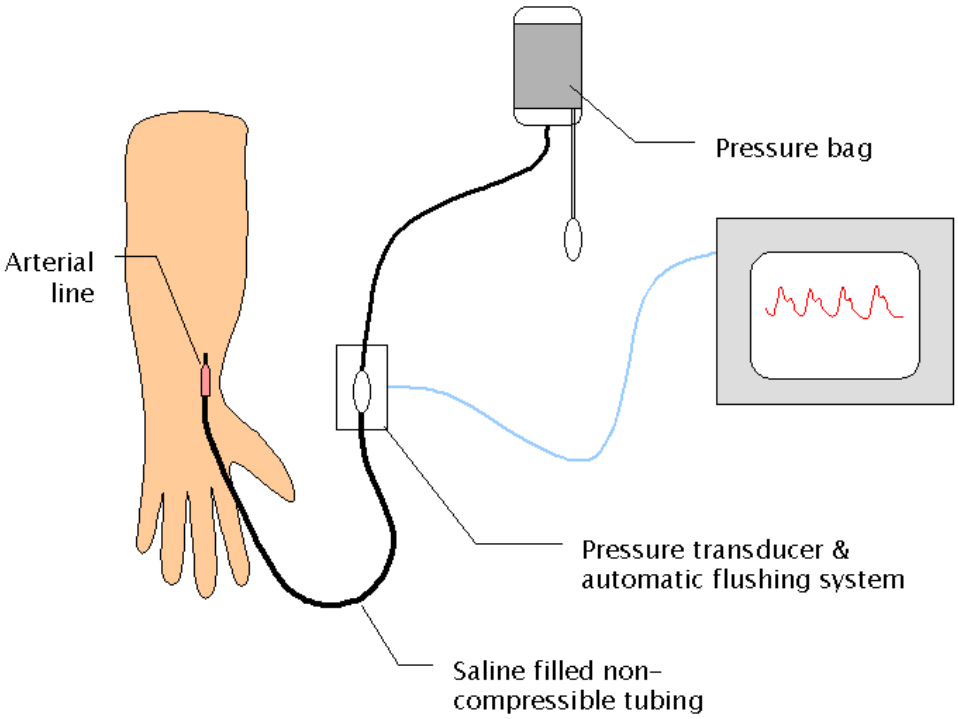

Figure 3-3: Arterial blood pressure measurement. An intra-arterial catheter (aka. arterial line, or in short A-line) is inserted into the patient's radial artery and is connected to a pressure bag through a tube filled with a saline solution, which contains a pressure transducer that records the ABP. Source of the figure: Vaughan et al. (2011).

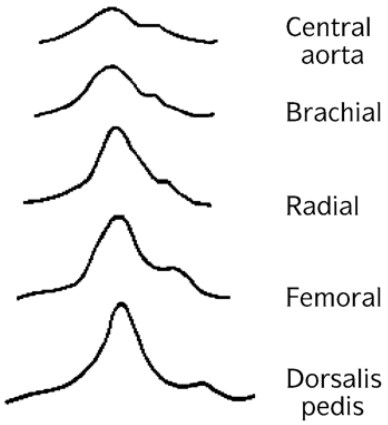

Figure 3-4: Impact of the measurement location on the blood pressure values. In MIMIC ABP, the blood pressure is measured fron one of the radial arteries. MIMIC also contains the blood pressure measurements from other locations, such as the femoral artery. Source of the figure: McGhee and Bridges (2002).

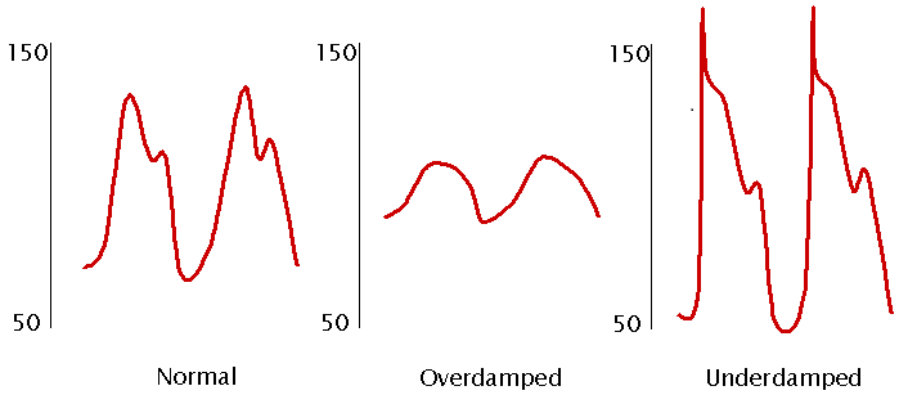

Figure 3-5: Impact of the damping degree on the blood pressure measurements: choosing the right damping degree is important to have a well-shaped signal. Source: Gomersall (2014).

3.3 Beat onset detection

We use the open-source software WFDB (WaveForm DataBase) developed by Moody et al. (2001) to detect the beat onsets. Figure 3-6 shows the output of the beat detection for twenty beats.

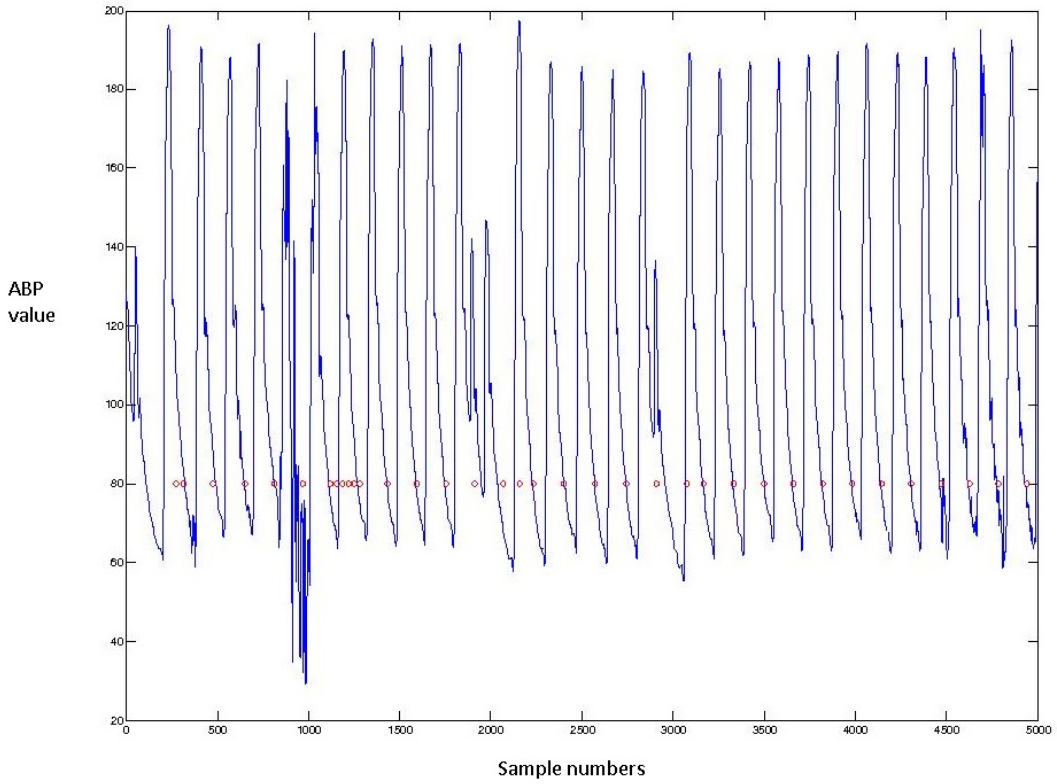

Figure 3-6: Beat onset detection. The blue curve represents the ABP signal; the red circles correspond to the beginning of each beat as the detected by WFDB.

As the signal is sometimes too noisy and the beat onset detection algorithm is not flawless, we marked each beat as being valid or invalid by using a beat validation heuristic, which is detailed in Waldin (2013b) and Sun et al. (2006). The heuristic consists in a set of 9 rules that defines thresholds for a few properties of a beat, such as *"if the pulse pressure less than 20 mmHg, then the beat is invalid"*.

Furthermore, the recording of the ABP signal sometimes contains jumps, that is to say that the signal was not recorded for a certain lapse of time. We can detect such jumps as each ABP sample in MIMIC has a timestamp. We therefore add a third

type of beat in addition to valid and invalid beats: jumps.

Figure 3-7 shows the number of valid beats for each patient. Figure 3-8 presents the percentage of valid beats for each patient. Figure 3-9 shows the time elapsed between two consecutive jumps for all the measurements we have.

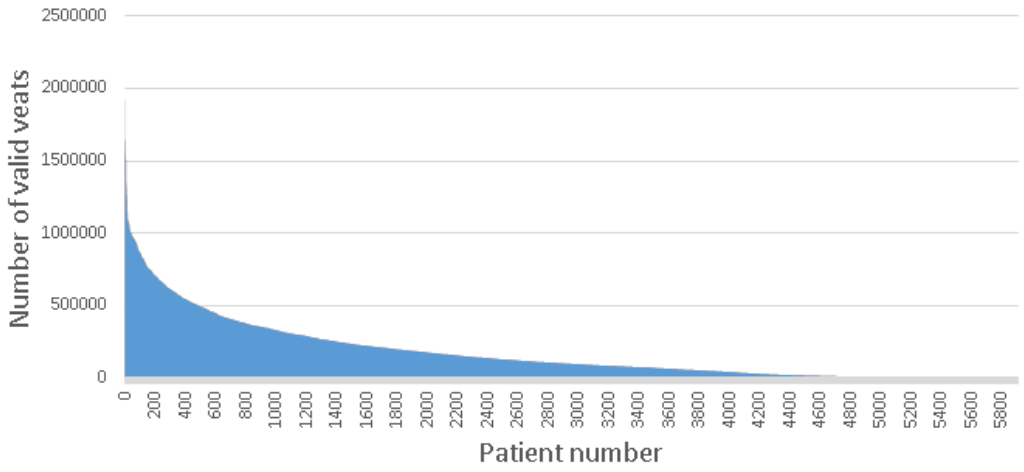

Figure 3-7: Number of valid beats per patient. We observe that some patients have a very small amount of valid beats: we discard those patients' data in the rest of this work.

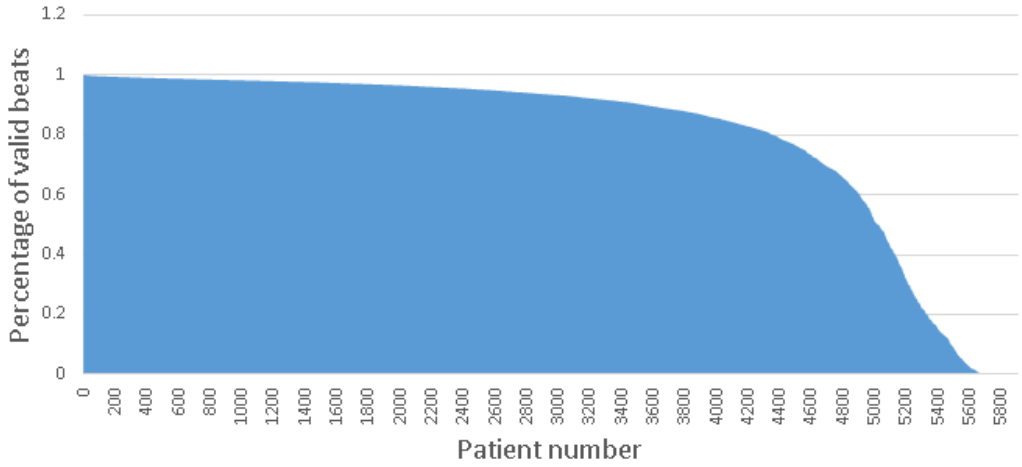

Figure 3-8: Percentage of valid beats per patient. We see that the vast majority of patient has over 80% of valid beats, but we also notice that a few patients had a very low number of valid beats. We discard those patients' data in the rest of this work.

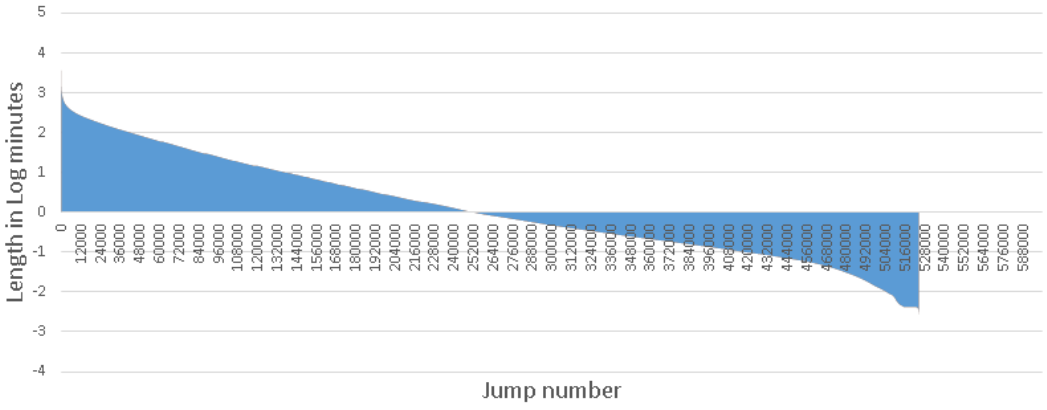

Figure 3-9: Duration of each ABP segment, i.e. time elapsed between two jumps. The abscissa represent the segment number, sorted by length. The ordinate is expressed in a logarithmic scale and represents the length of the segment.

3.4 Levels of noise

As in any data set, it is useful to carefully analyze the noise that affects our data. The dataset we have formed so far contains several layers of noise:

- The raw ABP signals are noisy. We have to keep in mind that the data was recorded in ICU where patients are typically in a critical condition and the medical personnel is under pressure to improve the patient's condition. For example the patient might move during the recording, which can perturb the ABP recording. By the same token, the nurse might not immediately see that the sensor is not working properly. Section 3.2 described the difficulty of measuring the ABP.

- The beat onset detection algorithm generates some noise as it sometimes fails to properly detect the beat. This might be caused by the first layer of noise, i.e. noise in the raw ABP sample values. Other factors might play a role: for instance, ventricular extrasystoles (aka. premature ventricular complexes, PVCs, or ventricular premature beats) might occur and perturb the beat onset detection. Kennedy et al. (1985) demonstrated that frequent (>60/h or 1/min) and complex extrasystoles could occur in apparently healthy subjects, with an estimated prevalence of 1-4%. They are even more likely to occur when a patient is in the ICU.

- Patient identification: some patient's records actually contain the record of several patients, and conversely a patient might be identified as being several patients in case he was admitted to the ICU several times.

As such we can regard this database as a low signal-to-ratio (SNR) database. We hope that the size of the dataset help compensate for the lack of clear signal, but this certainly represents an important challenge for our work.

Chapter 4

The prediction problem

This chapter presents the prediction problem that we will study using BeatDB, based on the MIMIC data set.

4.1 Acute hypotensive episode (AHE)

The blood pressure of a patient is a critical information in an ICU setting. Abnormal blood pressure level can be life-threatening, and might necessitate an immediate intervention from a nurse or a physician. We try to predict the occurrence of an acute hypotensive episode (AHE), which means that the blood pressure stays too low for too long. Left untreated, such episodes may result in irreversible organ damage and death. Timely and appropriate interventions can reduce these risks. For this work we focus on one signal only, the arterial blood pressure, which is both the input and the output our prediction problem.

As the definition of an AHE differs between physicians, we will make it parametrizable. An AHE takes place when, for some time window, with a minimum frequency (percentage), the mean arterial pressure (MAP) dips below a threshold (in mmHg). The MAP is the mean value of the blood pressure during one beat. For example, one definition of an AHE could be an event when 90% of MAP values in a 30 minute

window dip below 60 mmHg. We see that the definition of an AHE depends on three parameters:

1. the time window we consider,
2. the threshold above which we consider that the MAP is too low,
3. the percentage of beats whose MAP is too low.

Given the presence of noise, a fourth parameter is the minimum percentage of valid beats.

4.2 Objectives

The experiments of this chapter aims at answering the following questions using BeatDB:

- How does the condition threshold influence the prediction accuracy? (Section 4.3)
- To what extent will the features listed in Section 4.4 help predict AHEs? (Section 4.5)
- How does varying the lag and the lead impact the prediction quality? (Section 4.5)

4.3 Condition

Figure 4-1 shows the impact of the MAP threshold parameter on the number of patients that experienced AHE. Figure 4-2 illustrates how it impacts the number of AHE cases that are identified. Figure 4-3 shows how it changes the case (AHE) to control (non AHE) ratio.

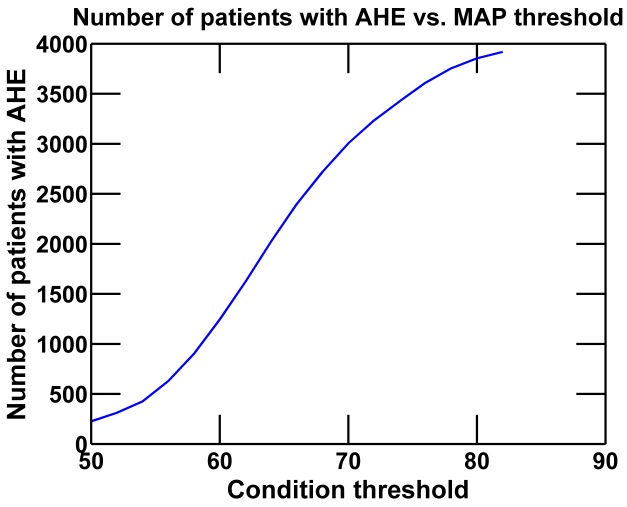

Figure 4-1: Number of patients with AHE events as we change the MAP threshold in the AHE event definition. The higher the threshold, the more patients with AHE events there are.

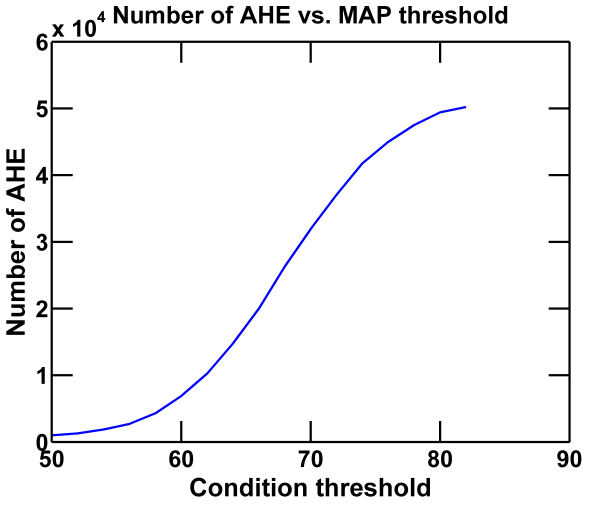

Figure 4-2: Total number of AHE cases present as we change the MAP threshold in the AHE event definition. The higher the threshold, the more balanced the data becomes.

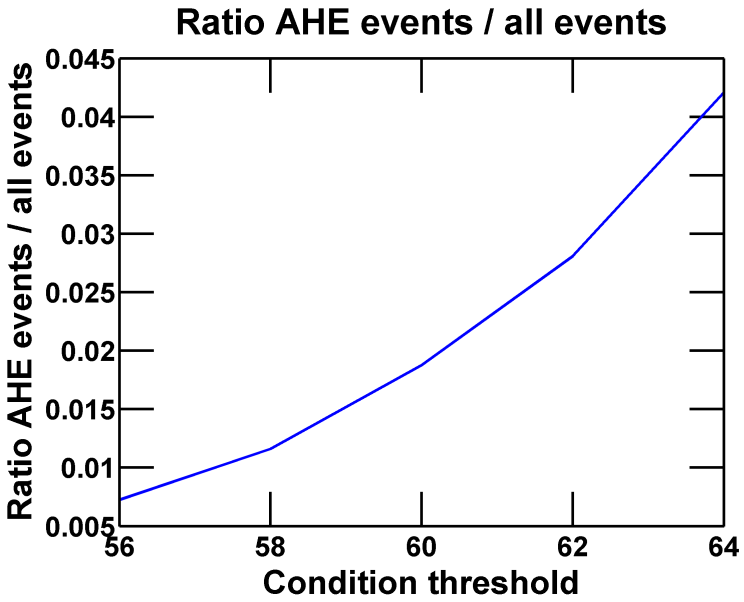

Figure 4-3: Balance between the AHE and non-AHE events as the MAP threshold changes. The higher the threshold, the fewer AHE events there are.

4.4 Features

As mentioned in Chapter 2, BeatDB integrates a storage space for beat-level features. We detail in this section the 14 features we used. In Chapter 5 we will develop a new set of features based on wavelets.

Since the ABP signal is sampled at 125 Hz, and a beat typically lasts around 1 second, one beat contain around 125 samples, each sample value being a simple floating-point number. A feature is a function that maps a beat's samples into a single floating-point number.

Below is the list of features we used. Let $x_1, x_2, \cdots, x_n$ be the samples of one beat. n is the number of samples in one beat, and is typically around 50 and 150, given that one beat lasts around one second and the blood pressure is sampled at 125 Hz, meaning that we have 125 samples for each second. Let $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ (mean), $\mu_n = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^n$ (n^{th} moment about the mean, aka. n^{th} central moment)), and $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(x_i - \mu)^2} = \sqrt{\mu_2}$ (standard deviation).

1. Root-mean-square (aka. quadratic mean). It measures the magnitude of the beat samples, and is defined as $x_{\text{rms}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}$

2. Kurtosis. It measures of how outlier-prone the beat samples are, or in other words degree of peakedness of the beat sample distribution. There exist several variants of kurtosis: we use the the kurtosis proper, which is defined as $x_{\text{kurtosis}} = \frac{\mu_4}{\sigma^4}$ (Abramowitz and Stegun, 1972).

3. Skewness. It measures the asymmetry of the beat samples around the beat sample mean. There exist several variants of skewness: we use the "main" one, which is defined as $x_{\text{skewness}} = \frac{\mu_3}{\sigma^3}$

4. Systolic blood pressure. As presented in Figure 3-2 it is the maximum sample value in the beat: $x_{\text{systole}} = \max_{1 \le i \le n} x_i$.

5. Diastolic blood pressure. As presented in Figure 3-2 it is the maximum sample value in the beat: $x_{\text{diastole}} = \min_{1 \le i \le n} x_i$.

6. Pulse pressure. It is the difference between the systolic and diastolic pressure measures: $x_{\text{pulse}} = x_{\text{systole}} - x_{\text{diastole}}$.

7. Duration of each beat. It is the number of samples that a beat contains: $x_{\text{duration}} = n$.

8. Duration of the systole. The systole occupies one third of the beat (Gad, 2008), hence the formula $x_{\text{systole\_duration}} = \frac{n}{3}$.

9. Duration of the diastole. The diastole occupies two thirds of the beat (Gad, 2008), hence the formula $x_{\text{diastole\_duration}} = \frac{2n}{3}$

10. Pressure area during systole: $x_{\text{diastole\_duration}} = \sum_{i=1}^{\lceil x_{\text{systole\_duration}} \rceil} (x_i - x_{\text{diastole}})$

11. Standard deviation of signal. This corresponds to σ.

12. Crest factor (aka. square root of the peak-to-average ratio). The crest factor indicates how extreme the peaks are in a waveform. If it is equals to 1, it means that the signal has no peak. It is defined as $x_{\text{crest}} = \frac{x_{\text{systole}}}{x_{\text{rms}}}$.

13. Mean of signal. This corresponds to μ.

14. Mean arterial pressure (MAP). We compute the MAP based on the systolic and diastolic blood pressure values using the formula presented in Zheng et al. (2008) in the definition section: $x_{\text{map}} = \frac{x_{\text{systole}} + 2x_{\text{diastole}}}{3}$.

4.5 Results

In our experiment we use BeatDB with the 5 different aggregate functions (see Figure 2-4 regarding the use of aggregate functions in BeatDB) and 14 different per beat features, with 1-minute sub-windows in the lag, resulting in 70 features (5×14) per training exemplar. Then we prepare a data set for each parameter combination: 5 MAP thresholds, 4 lags and 6 leads, resulting in 120 combinations. On these, we execute a 10-fold cross-validation on our lab's private cloud using approximately 2 nodes with 24 VCPUs each for 48 hours. The parameters are summarized in Table

4.1. The results across different condition thresholds and prediction leads and lags can be viewed in Figure 5.

Parameter names	**Parameter choice**
Condition's window size	30 minutes
Condition's threshold	56, 58, 60, 62, 64 mmHg
Condition's frequency	90%
Condition's variable	MAP
Prediction's lag	10, 20, 30, 60 minutes
Prediction's lead	10, 20, 30, 60, 120, 180 minutes
Prediction's features	14 features
Sub-aggregation window	1 minute
Sub-aggregation function	Mean
Aggregation functions	Mean, standard deviation, kurtosis, skew, trend
Machine learning algorithm	Logistic regression
Evaluation metric	AUC of the ROC (aka. AUROC)

Table 4.1: Parameter for the AHE prediction.

Figure 4-4 shows the area under the ROC curve (AUROC) for different lead times, given maximum lag of 60 minutes for 5 different MAP thresholds. The AUROC drops as the lead time increases. The prediction problem becomes easier when a threshold of 56 mmHg is chosen as the average MAP. When this threshold increases, predicting AHE, given this data, becomes harder. This confirms expectations because 56 mmHg is an extreme threshold point and we would expect that the cohort of patients with such an extreme condition will be significantly be different then the rest of the patients. Figure 4-5 shows the AUROC when we change the *lag* and keep the *lead* at its minimum 10 minute duration. In this case, we see that the performance improves as we increase the lag or historical data taken into account. This is intuitive because a longer lag provides more signal to learn from.

Next, in order to explore a point solution on the ROC curve, we chose a true positive rate of 90% for AHE and evaluated the false positive rate. Figures 4-6 and 4-7 show the results for different lags and leads for different definitions of AHE. As in the previous results, more signal history helps prediction.

Interestingly, we notice on Figures 4-5 and 4-7 that when the lead becomes very high,

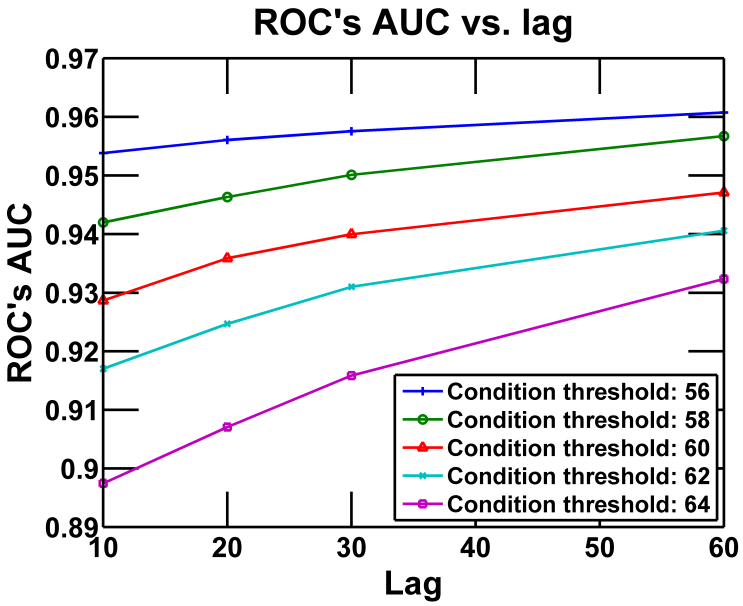

Figure 4-4: Impact of the lag on the AUROC with different condition thresholds. As seen in Section 4.1, the threshold is a value expressed in mmHg above which we consider that the MAP is too low. As expected, the longer the lag, i.e. the further we look into the history, the more accurate the prediction becomes, although we see that past 30 minutes increasing the lag doesn't matter much. We can also remark that as the condition threshold increases, the prediction AUROC decreases. This is due to the fact that increasing the condition threshold result in a larger number of AHE events, as we saw in Figure 4-2, which become harder to distinguish from the non-AHE events.

over 120, the AUROC or the FPR starts to increase. This is counter-intuitive because that would mean that the higher the lead, the more accurate the prediction is. In fact, this result is due to the fact that when the lead is very high, models start to overfit as the number of AHE events is getting scarce.

The results of this section were published in Dernoncourt et al. (2013c).

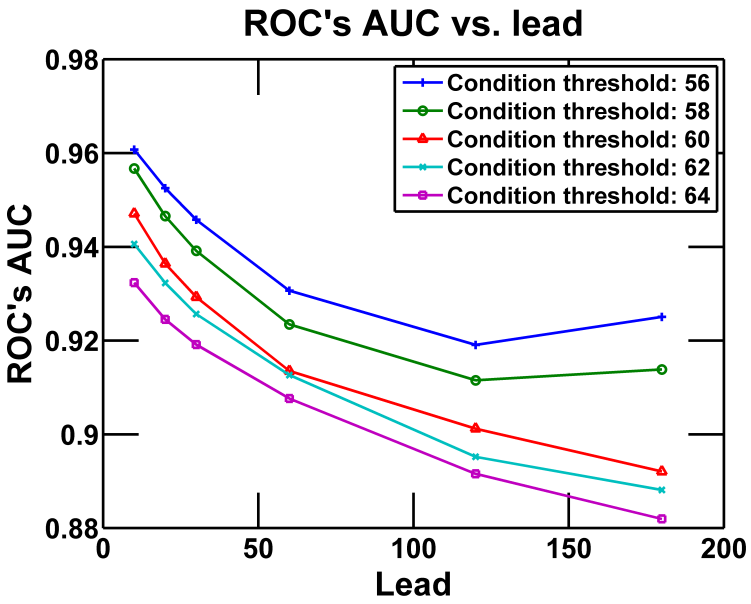

Figure 4-5: Impact of the lag on the AUROC with different condition thresholds. As in Figure 4-4, increasing the condition threshold result in a lower AUROC. Also, as expected, increasing the lead makes the prediction harder (i.e. lower AUROC), although we do see some increase when the lead becomes very high due to the scarcity of AHE events in those situations.

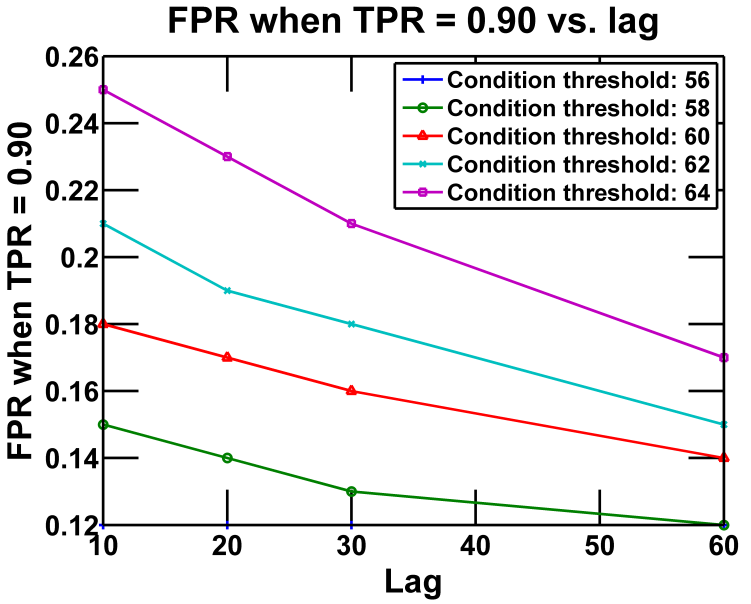

Figure 4-6: Impact of the lag on the FPR when $TPR = 0.9$. This figure is similar to Figure 4-4 but with a fixed TPR.

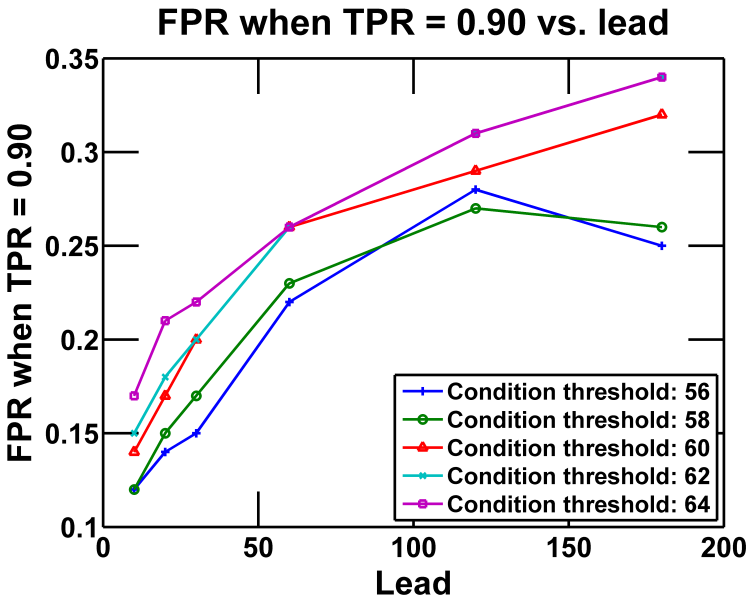

Figure 4-7: Impact of the lead on the FPR when $TPR = 0.9$. This figure is similar to Figure 4-5 but with a fixed TPR.

Chapter 5

Wavelets as features

In the previous chapter we used 14 features aggregated using 5 different statistics, with only 1 sub-window in the lag, and changed the lag, the lead as well as the condition threshold. For each parameter set we assemble the data set accordingly and use logistic regression to predict an AHE event. The quality of the prediction is accessed using the AUC of the ROC (aka. AUROC).

In this chapter we focus on one specific set of features: continuous wavelet transforms. Wavelets offer some interesting properties that might be useful to detect abnormalities in the beat signal, and therefore have the potential to drastically improve the prediction accuracy for some medical conditions. Table 4.1 summarizes all the parameters that were used for the experiment.

Continuous wavelet transforms present a specific challenge: a scale and a time shift have to be specified. For each wavelet transform we investigate, we try 10 different scales and 19 different time shifts in order to thoroughly explore the wavelet's ability to predict an AHE. Unlike the previous chapter, we do not change the condition threshold of the AHE: we fix it at 60, which is the most commonly used value. We use sub-windows of size a tenth of the lag, instead of 1-minute fixed-size sub-windows as in the previous experiment, but using only 1 statistic as the sub-aggregation function, the mean, and we do not use any aggregation function. As a result, we will have 10

final features (1 feature $\times$ 10 sub-windows $\times$ 1 sub-aggregation function). Lastly, we reduce the length of the lag and the lead in order to avoid the model to overfit, as we have seen in the previous chapter that high lag and lead cause a shortage of AHEs.

Table 5.1 summarizes all the parameters that we use in experiments presented in this chapter.

Parameter names	**Parameter choice**
Condition's window size	30 minutes
Condition's threshold	60 mmHg
Condition's frequency	90%
Condition's variable	MAP
Prediction's lag	$10, 20, 30, 40, 50, 60$ minutes
Prediction's lead	$10, 20, 30, 40, 50, 60$ minutes
Prediction's features	1 feature (one wavelet with a specific scale and time shift)
Sub-aggregation window	$\frac{1}{10}$ of the lag
Sub-aggregation function	Mean
Aggregation function	None
Machine learning algorithm	Logistic regression
Evaluation metric	AUC of the ROC (aka. AUROC)

Table 5.1: Parameter for the AHE prediction using wavelets.

5.1 Objectives

We try only one specific wavelet transform at a time to predict AHEs. In other words, we use just one feature: wavelet transform with a specific scale and time shift. We want to answer the following questions:

- Which wavelet transform can best help predict the event? (Section 5.4.1)
- What is the best scale and time shift for each wavelet transform, i.e. scale and time shift that has the highest predictive ability? (Section 5.4.1)
- Does the best scale and time shift for each wavelet transform depend on the lag or lead? (Section 5.4.1)

- Does combining the wavelet features with the 14 features we used in the previous chapter have any additional discriminatory power for the prediction? (Section 5.4.2)
- Does a larger data set size lead to a more accurate prediction? (Section 5.4.3)

To find the best scale and time shift with different lag and lead, we perform a grid search for each wavelet transform (i.e. we exhaustively enumerate all possible combinations of parameters), use a logistic regression and compute the AUC of the ROC (aka. AUROC) as the evaluation metric. As in the previous chapter, we use the entire MIMIC Waveform Database, which contains over 1 billion beat and 5000 patients. For each wavelet transform, all 10 scales and 19 time shifts are pre-computed: we do not compute them on the fly because computing all scales and time shifts at once is much more efficient than computing specific scale and time shift one by one. Each extracted wavelet takes around 300 GB worth of data when compressed. Performing a grid search for each wavelet transform is a costly process: we will optimize it in Chapter 6 by using a Gaussian process to explore the space.

We first present motivation behind the use of wavelets, then thoroughly apply it as features for our prediction problem and present the results we obtain.

5.2 Wavelets

We use wavelets to perform a continuous wavelet transform on a signal. A wavelet is a brief oscillation with the shape of a wave, hence its name. Figure 5-1 shows three different wavelets.

Continuous wavelet transforms are a generalization of Fourier transforms. They transform signals between time (or spatial) domain and frequency domain, as shown in Figure 5-2. The major difference between continuous wavelet transforms and Fourier transforms is that the former is localized in both time and frequency while the Fourier transform is only localized in frequency. Subsequently unlike Fourier transforms

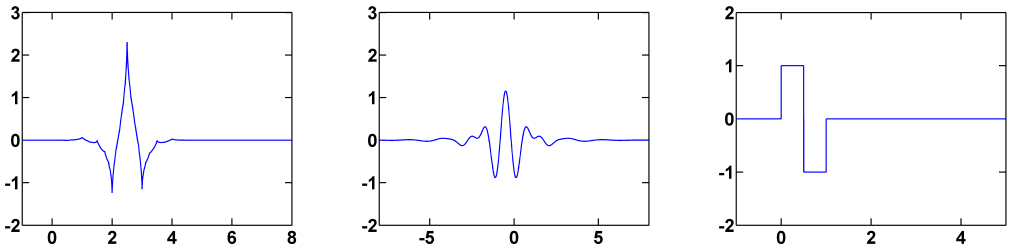

Figure 5-1: Examples of wavelets. From right to left: Coiflets-1, Meyer, and Haar wavelets.

wavelets have the ability to unveil location-specific features within the signal (Addison, 2005), which can be vital to detect tiny abnormalities in specific parts of a beat. Furthermore there exist many wavelet functions, which increases the chance to spot useful signals for the prediction problem that we try to solve. Wavelets have already been used for ECG (Li et al., 1995) and many other medical applications (Unser and Aldroubi, 1996).

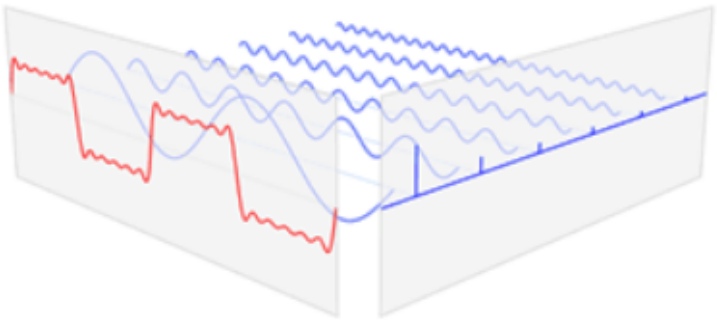

Figure 5-2: Relation between the function's time domain, shown in red, to the function's frequency domain, shown in blue. Source: Wikipedia.

The core idea behind continuous wavelet transform is to use convolutions to quantify how similar the given signal is compared with the given wavelet, like with the Fourier transform. However, unlike the Fourier transform, we need to specify both a scale parameter a (strictly positive real, aka. dilation parameter), and a time shift, b (any real, aka. position or translation). The resulting formula is the following:

$$X(a, b; f, \psi) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) \overline{\psi} \left(\frac{t - b}{a} \right) dt$$

where:

- ψ is the chosen wavelet,
- f is the signal to be analyzed.

As a result, we can define a single feature on a beat by specifying one wavelet, one scale and one time shift. To put it otherwise, choosing one wavelet, one scale and one timeshift allows us to extract one single value from one beat, which we can try using in our prediction problem. Figure 5-3 shows the Symlet-2 wavelet with different scales and time shifts.

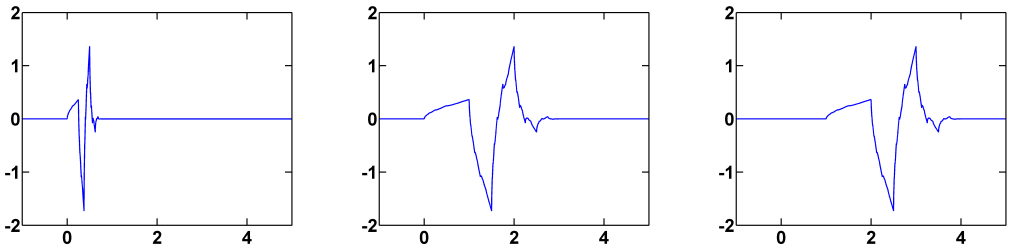

Figure 5-3: The Symlet-2 wavelet with different scales and time shifts:
- the leftmost figure shows the Symlet-2 wavelet with scale 0.25 and time shift 0,
- the middle figure shows the Symlet-2 wavelet with scale 1 and time shift 0,
- the rightmost figure shows the Symlet-2 wavelet with scale 1 and time shift 1.

5.3 Correlation between wavelets

Among the 87 wavelet transforms, some are highly correlated. Figures 5-4 and 5-5 show the correlation on some random data between all wavelet transforms for a given scale using the Pearson's linear correlation coefficient. Figures 5-6, 5-7 and 5-8 show the correlation between different scales of the same wavelet transform. The code and more such figures are available online [1].

[1] Correlation between all wavelets: http://stackoverflow.com/q/24394549/395857

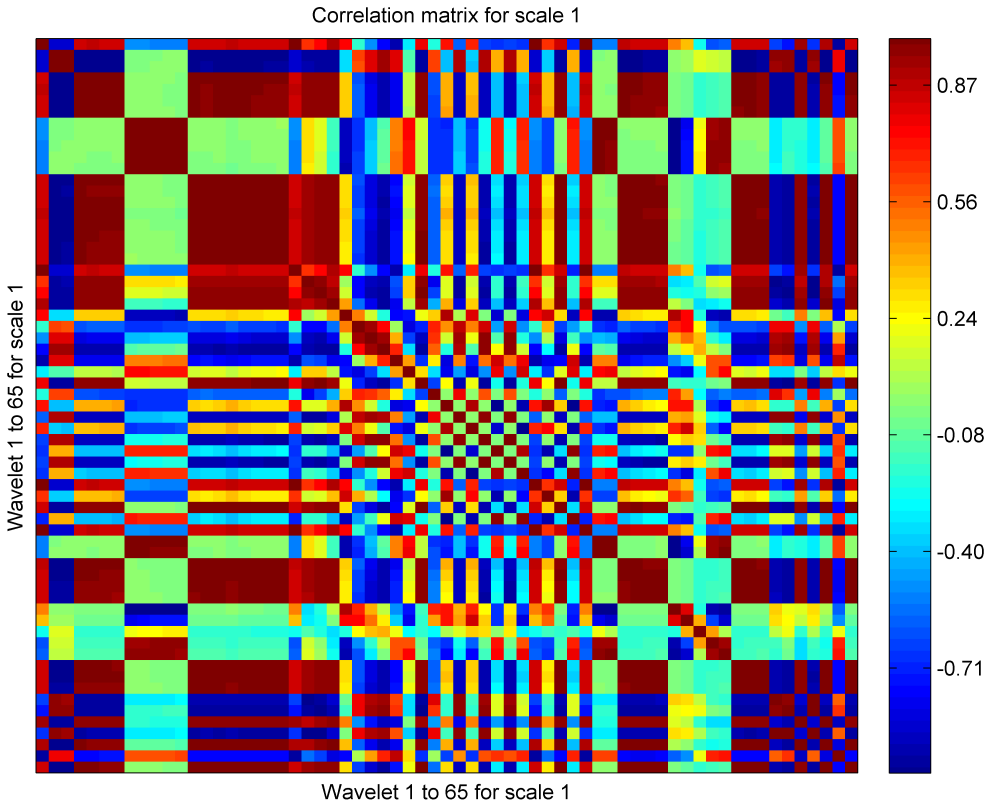

Figure 5-4: Correlation between the 65 different non-complex wavelets that are present in the Matlab's Wavelet Toolbox. In this figure, we computed the correlation for the first scale using a random signal.

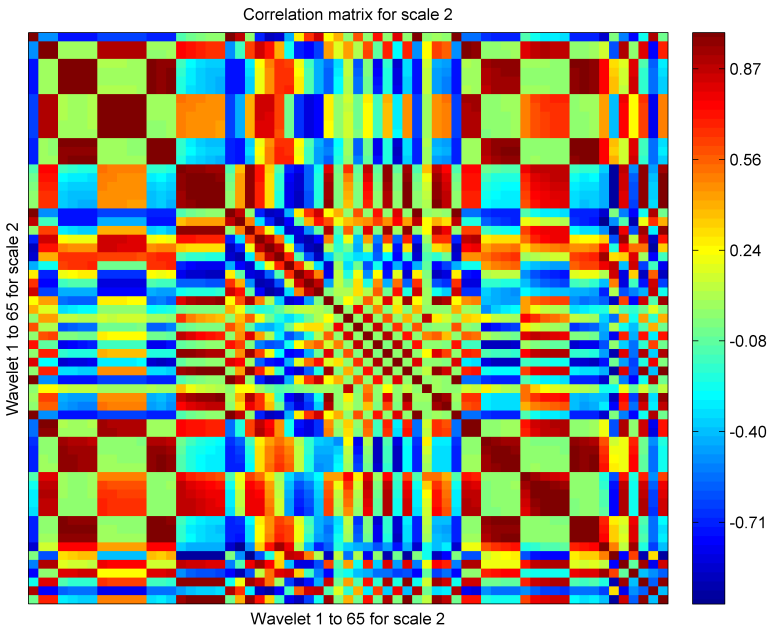

Figure 5-5: Correlation between the 65 different non-complex wavelets, as in Figure 5-4. Here we computed the correlation for the second scale. The main patterns are similar as in Figure 5-4 but the numbers are sometimes significantly different.

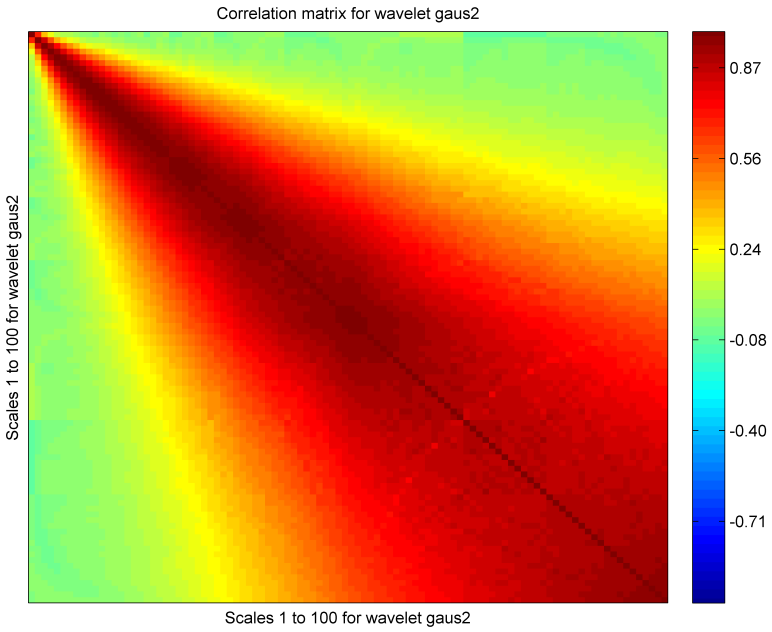

Figure 5-6: Correlation between the first 100 scales of the Gaussian-2 wavelet. We see that the closer two scales are, the more they are correlated.

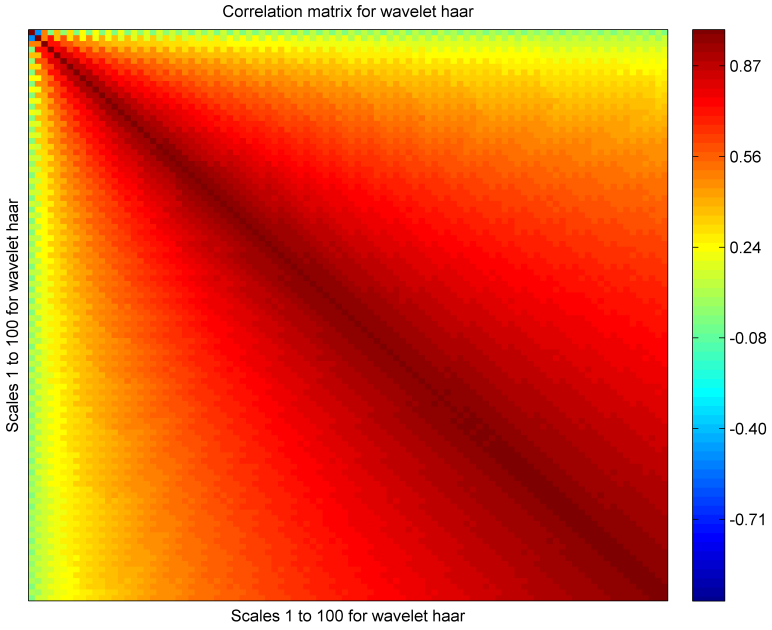

Figure 5-7: Correlation between the first 100 scales of the Haar wavelet. As with the Gaussian-2 wavelet in Figure 5-6, the closer two scales are, the more they are correlated.

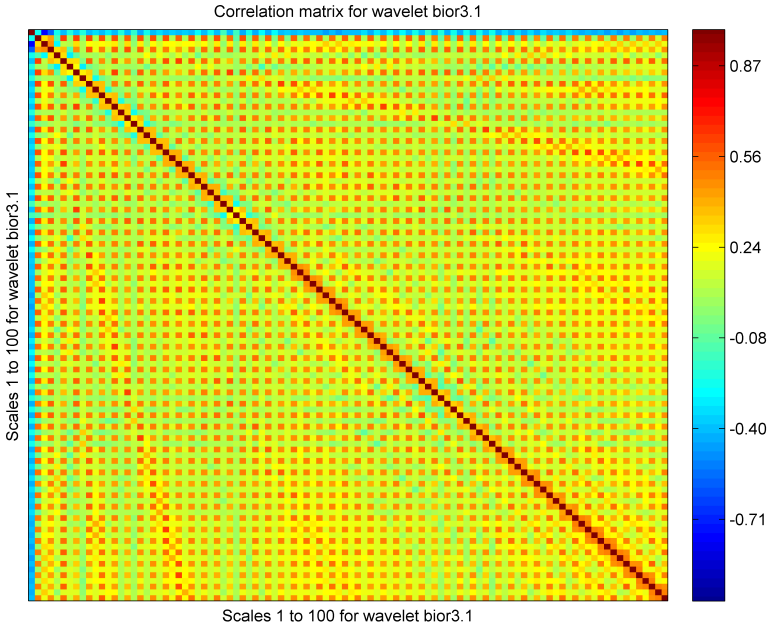

Figure 5-8: Correlation between the first 100 scales of the bior3.1 wavelet. The heatmap looks very different from the Gaussian-2 (Figure 5-6) and Haar (Figure 5-7) wavelets.

5.4 Experiments

5.4.1 Prediction experiment

When computing the continuous wavelet transforms of a beat, three parameters need to be decided:

1. the wavelet transform to be used;
2. the scale to apply on the wavelet transform;
3. the time shift we are interested in, i.e. where in the beat we want to compute the wavelet, since wavelets are located both in time and frequency.

We try to find the combination of those three parameters that yields the most accurate prediction. In other words, we predict the AHE using as the only feature one wavelet with one scale and one time shift.

In order to find the best three parameters, we explore the first 10 scales for 4 different wavelet transforms, and divide each beat into 19 different time shifts. As in the previous chapter, we use the AUROC as the metric for the prediction accuracy.

- Figure 5-9 presents a heat map of the AUROCs for all 10 scales and 19 time shifts using the Symlet-2 wavelet transform. The best AUROC is 0.72.
- Figure 5-10 shows the same for the Gaussian-2 wavelet transform. The best AUROC is 0.78.
- Figure 5-10 shows the same for the Haar wavelet transform. The best AUROC is 0.70.
- Figure 5-12 shows the same for the Bior 3.5 wavelet transform. The best AUROC is 0.76.

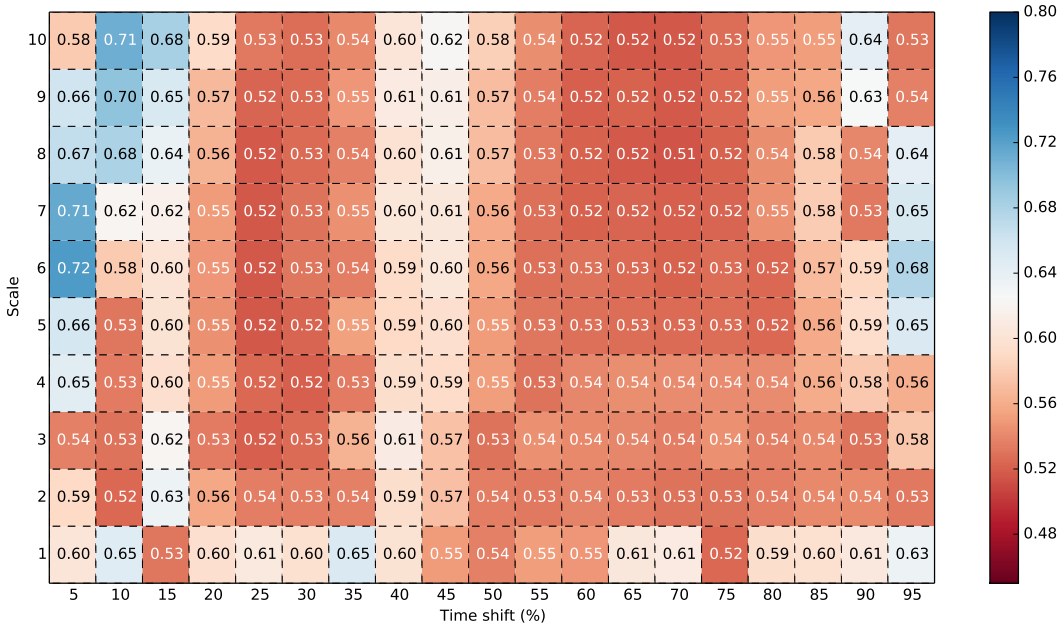

Figure 5-9: Heat map of the AUROCs for all 10 scales and 19 timeshifts using the *Symlet-2* wavelet transform using a lag of 10 minutes, and a lead of 10 minutes. The highest AUROC is 0.72 and is achieved with scale 6 and timeshift 5.

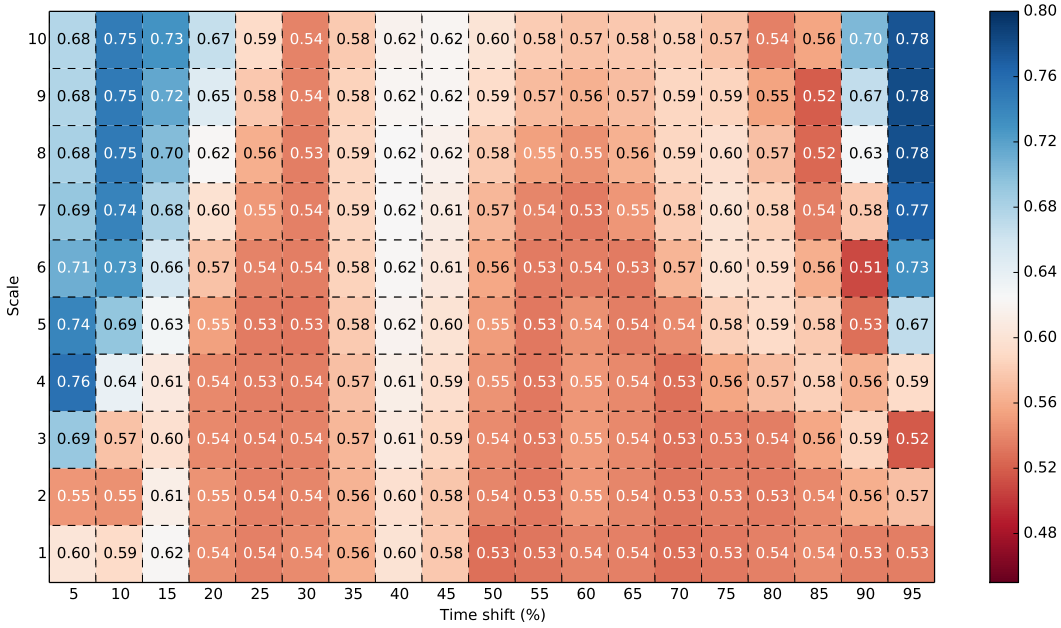

Figure 5-10: Heat map of the AUROCs for all 10 scales and 19 timeshifts using the *Gaussian-2* wavelet transform using a lag of 10 minutes, and a lead of 10 minutes. The highest AUROC is 0.78 and is achieved with scale 9 and timeshift 95.

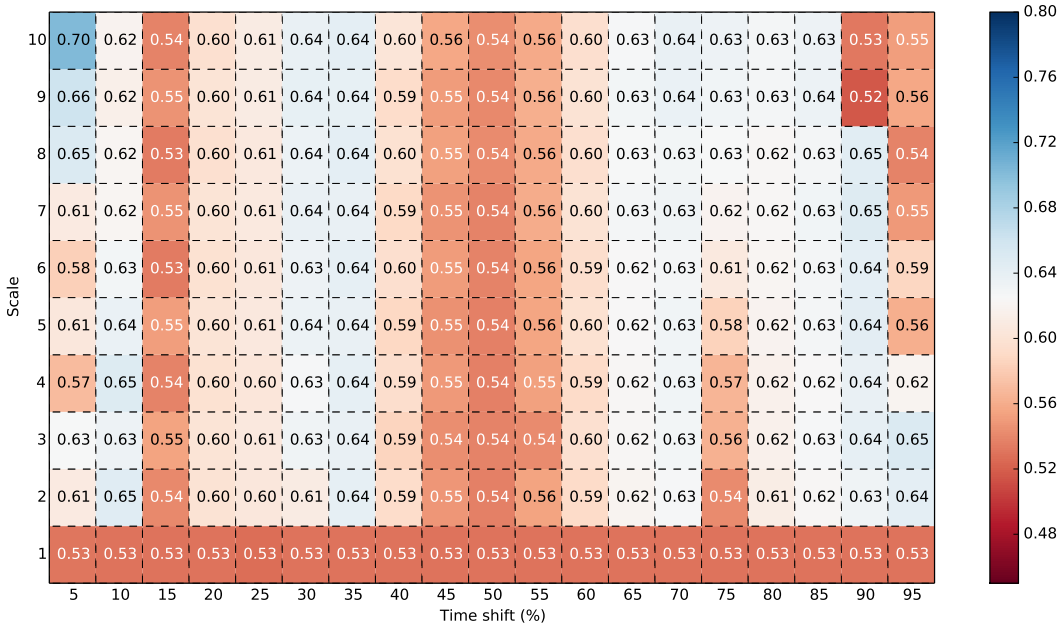

Figure 5-11: Heat map of the AUROCs for all 10 scales and 19 timeshifts using the *Haar* wavelet transform using a lag of 10 minutes, and a lead of 10 minutes. The highest AUROC is 0.70 and is achieved with scale 10 and timeshift 5.

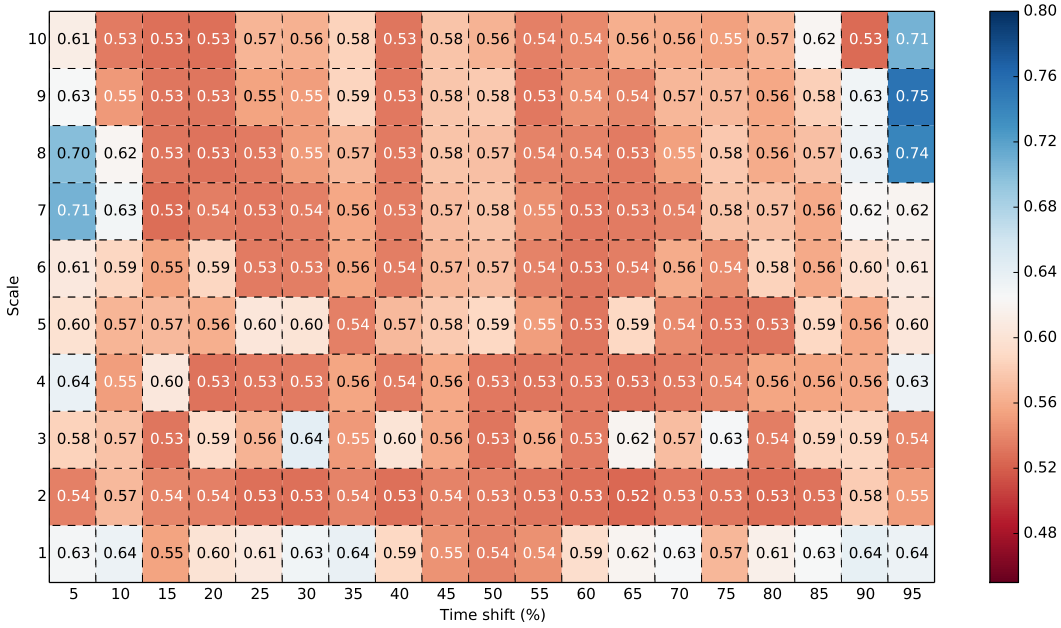

Figure 5-12: Heat map of the AUROCs for all 10 scales and 19 timeshifts using the *Bior 3.5* wavelet transform using a lag of 10 minutes, and a lead of 10 minutes. The highest AUROC is 0.75 and is achieved with scale 9 and timeshift 95.

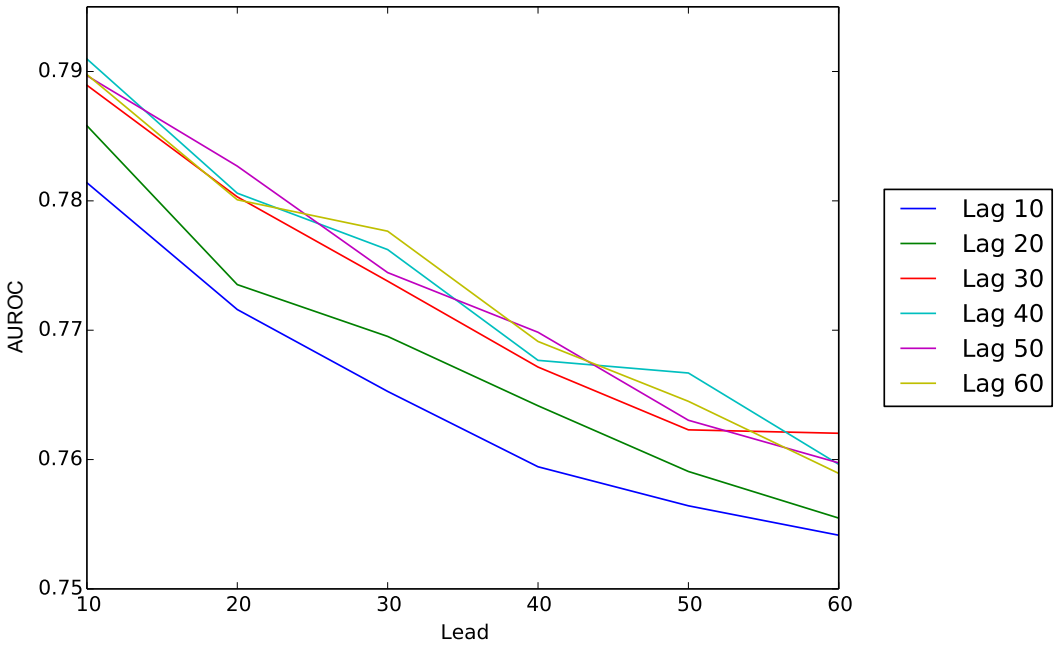

Figure 5-13: Influence of the lead on the AUROC for the Gaussian-2 wavelet. For each pair of lag and leag, we take the maximum AUROC achieved among all 10 scales and 19 timeshifts. Increasing the leads reduces the AUROC, which is not surprising as a higher lead means that the prediction is made further in time. Increasing the lag from 10 to 30 minutes improves the AUROC, but once 30 minutes is reached increasing the lag doesn't further improve the AUROC.

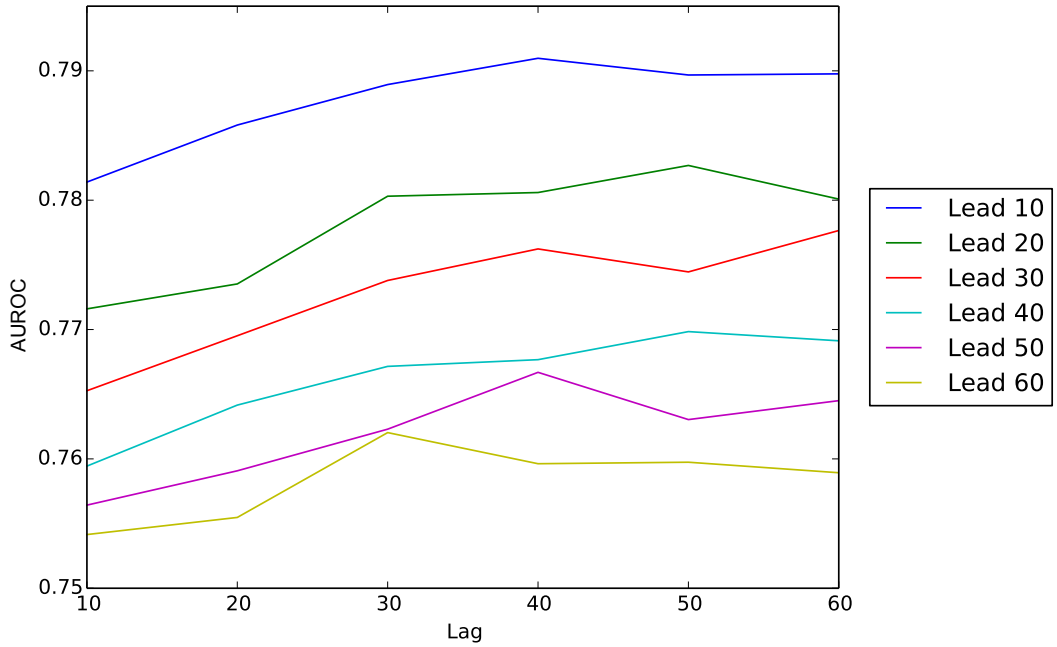

Figure 5-14: Influence of the lag on the AUROC for the Gaussian-2 wavelet. The graph is the transpose of Figure 5-13, so the same conclusions apply.

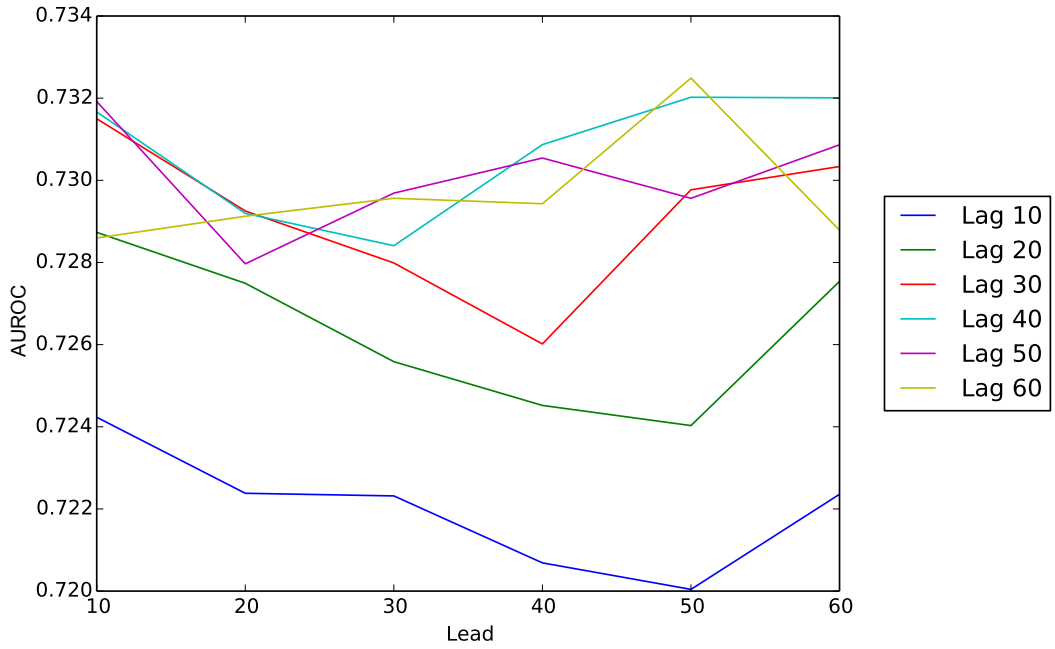

Figure 5-15: Influence of the lead on the AUROC for the Symlet-2 wavelet. This is the same graph as Figure 5-13 but for the Symlet-2 wavelet.

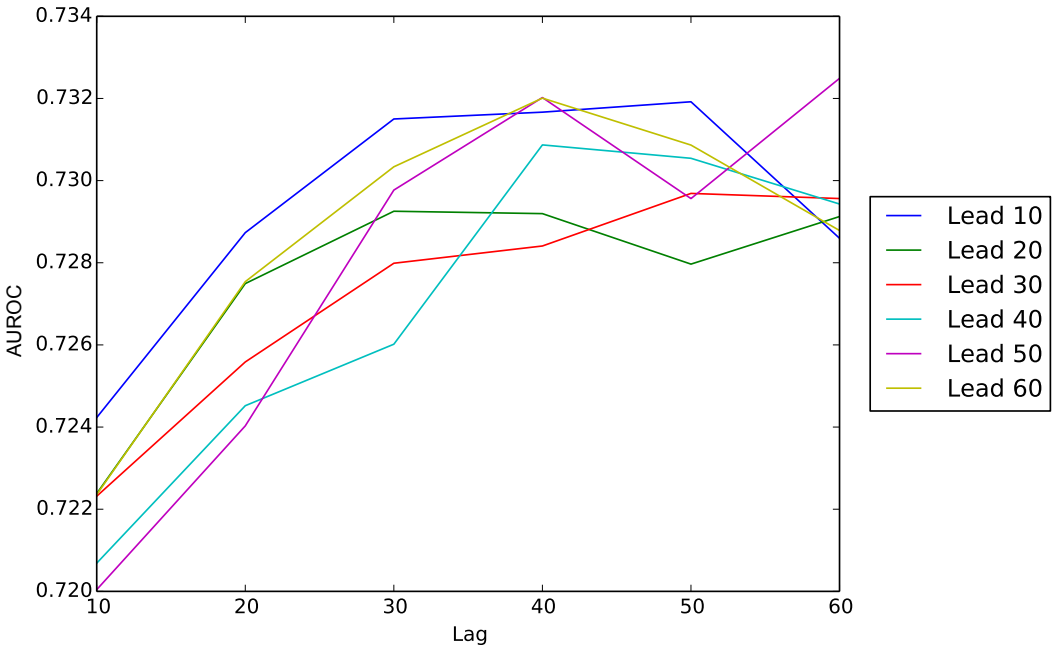

Figure 5-16: Influence of the lag on the AUROC for the Symlet-2 wavelet. This is the same graph as Figure 5-14 but for the Symlet-2 wavelet.

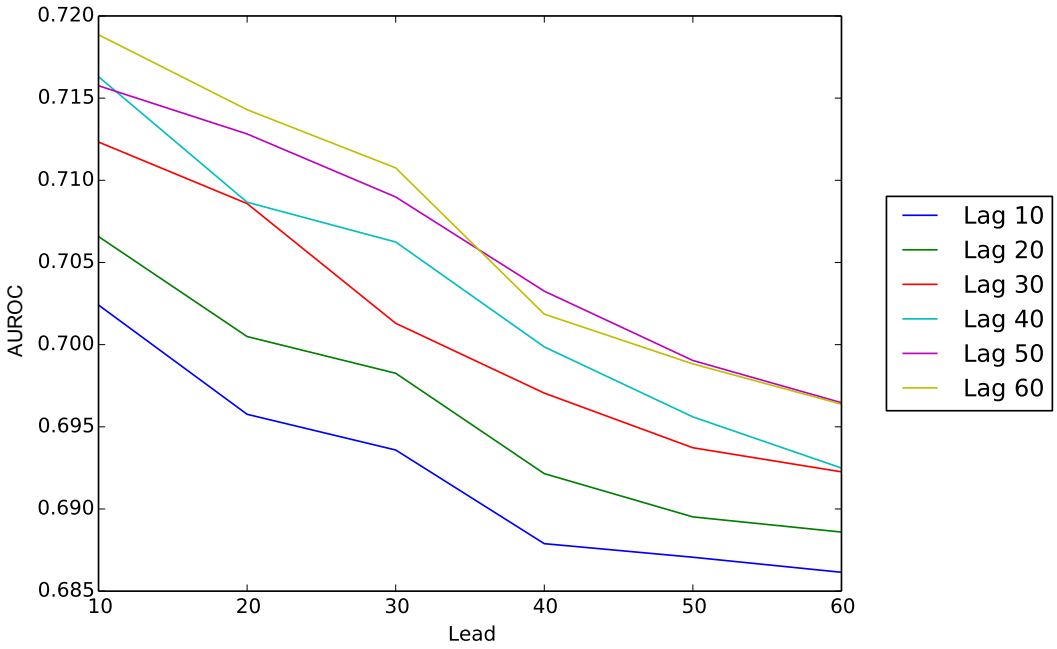

Figure 5-17: Influence of the lead on the AUROC for the Haar wavelet. This is the same graph as Figure 5-13 but for the Symlet-2 wavelet.

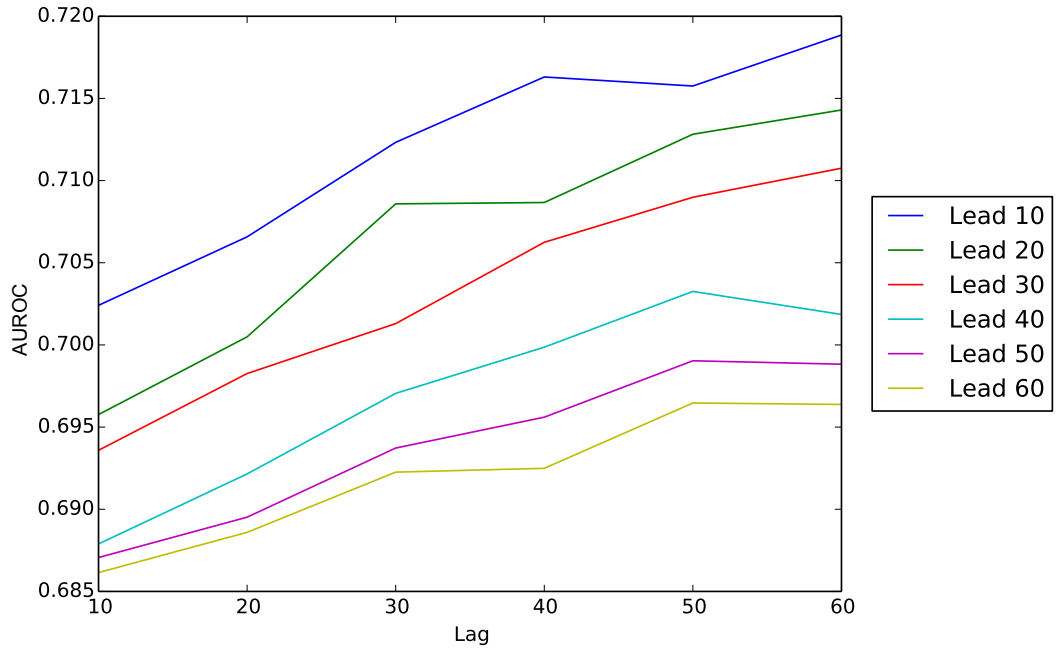

Figure 5-18: Influence of the lag on the AUROC for the Haar wavelet. This is the same graph as Figure 5-14 but for the Symlet-2 wavelet.

From those experiments we see that:

- The choice of the wavelet transform has an important impact on the prediction's AUROC. Amongst the 4 wavelet transforms that we have tried, the Gaussian-2 wavelet has the highest AUROC when predicting AHEs.
- It is critical to choose the right scale and the time shift. Interestingly, the best scales and time shifts stay the same when we change of the lag and lead. Across wavelet transforms we can see some similarities between the best scales and time shifts.

5.4.2 Wavelets in addition to the other 14 features

Now that we have found the best scale, time shift, lag and lead for each wavelet transform, we investigate whether these wavelet features can improve the prediction quality when added to the 14 features used in the previous chapter.

Table 5.2 summarizes the results when wavelet features are added to the 14 previous features. Our results show that adding a wavelet leads to a negligible increase of the AUROC, and a slight increase of the FPR when we fix the TPR to 0.90. This result is not surprising as the 14 previous features had very high AUROC.

Interestingly, when we only consider MAP as a feature and add a wavelet to it, the increase of the AUROC is more significant, 0.01, which is not negligible as past 0.90 it is generally hard to further increase the AUROC.

As a last observation, even though the Gaussian-2 wavelet achieves a higher AUROC than the Haar and Haar wavelet when considered as a single feature, when added with the MAP feature it does not achieve a higher AUROC.

	Features	Number of features	AUROC	FPR when TPR = 0.90
(1)	14 initial features using the 5 aggregation functions	70	0.9523	0.14
(2)	14 initial features using the mean aggregation function	14	0.9409	0.19
(3)	MAP feature using the 5 aggregation functions	5	0.9092	0.37
(4)	Gaussian-2 wavelet with scale 9 and time shift 95	10	0.7897	0.61
(5)	Haar wavelet with scale 10 and time shift 5	10	0.7187	0.63
(6)	Symlet-2 wavelet with scale 6 and time shift 5	10	0.7286	0.61
	(1) + (4)	80	0.9529	0.12
	(1) + (5)	80	0.9528	0.12
	(1) + (6)	80	0.9525	0.12
	(2) + (4)	24	0.9423	0.15
	(2) + (5)	24	0.9430	0.15
	(2) + (6)	24	0.9411	0.15
	(3) + (4)	15	0.9156	0.24
	(3) + (5)	15	0.9123	0.25
	(3) + (6)	15	0.9170	0.25

Table 5.2: Wavelets in addition to the other 14 features

5.4.3 Impact of the size of the data set on the prediction accuracy

Carrying out the experiments for 5000 patients is computationally expensive. It is therefore worthwhile to investigate whether this large number of patients is justified. For that purpose, we use the Gaussian-2 wavelet as a feature and explore the first 10 scales, 19 different time shifts, 6 different lags and 6 different leads as we have done in the previous section. We perform this experiment with 1000, 2500 and 5000 patients, and compare the best AUROC found. Figure 5-19 summarizes the findings: increasing the number of patients does have a positive impact on the AUROC.

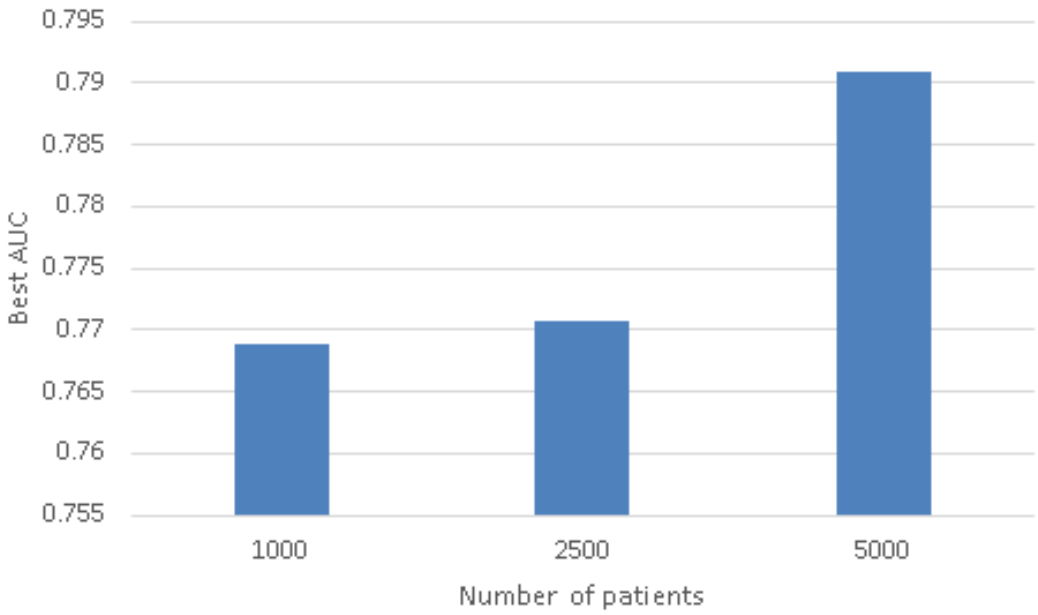

Figure 5-19: Influence of the data set size on the AUROC for the Gaussian-2 wavelet.

5.4.4 Computational cost

Each experiment, i.e. each time we compute the AUROC for a given wavelet transform, scale, time shift, lag and lead, it takes around 1 hour on a healthy 4-core machine with average specifications and with a fast network connection to BeatDB data. As we have seen in the previous section, for one wavelet we perform $10 \times 19 \times 6 \times 6 = 6840$ ex-

periments since we try 10 different scales, 19 different time shifts, 6 different lags and 6 different leads. This amounts to 6840 hours (150 days) of computation to explore one wavelet. In the next chapter we will use Gaussian Processes to avoid computing exhaustively all parameter sets, but instead search for the best parameters in a more clever fashion.

As indicated in Section 2.8, we use an OpenStack cluster and we dedicate around 100 4-core instances to explore wavelet, the exact number of instances depending on the resource availability.

Chapter 6

Gaussian process for parameter optimization

As we have seen in the previous section, using a wavelet as a feature requires us to go through different parameters such as the scale, the time shift, the lag and the lead so as to find the highest AUROC. Exhaustively searching through those parameters is computationally costly. For example, so far we have used 10 different scales, 19 different time shifts, 6 different lags and 6 different leads, which means we have to perform $10 \times 19 \times 6 \times 6 = 6840$ experiments to find the best set of parameters. We would like to cut down this number.

One way could be to simply randomly select those parameters, perform a given amount of experiments, e.g. 200, and declare that the best AUROC found in those 200 experiments is a reasonable approximation of the best AUROC we could find by trying all parameters.

However, the heat maps presented in Figures 5-9 and 5-10 reveal some patterns that we would like to take advantage of to converge faster to the best AUROC. For that purpose we use Gaussian process regression, inspired by Snoek et al. (2012) and Drevo (2014).

In this chapter we aim at answering the four following questions:

- How much computational effort can using a Gaussian Process save? (Sections 6.1, 6.2 and 6.3)
- What are the optimal Gaussian Process parameters? (Sections 6.1 and 6.2)
- How can we distribute a Gaussian Process over several instances? (Section 6.3)
- To what extent does distributing a Gaussian Process impact its convergence speed? (Section 6.3)

6.1 Choosing the kernel

Since each experiment with 5000 patients takes around 1 hour, we do not want to exhaustively compute the search space. Instead of performing a random search, we choose to model the parameter space with a Gaussian process. We investigate how much faster the Gaussian process finds the best combination of parameters in comparison with the random search, as well as what are the best hyper-parameters to use for the Gaussian process.

We use a Gaussian process regression as follows: we first compute the AUROC for 10 sets of parameters drawn randomly. Then, we set:

- x_{train} to be the 10 combinations of parameters randomly drawn.
- y_{train} to be the 10 computed AUROCs (one for each combination of parameters).
- x_{test} to be the set of all remaining combinations of parameters.

We compute y_{test} using Gaussian process regression as defined in Appendix E.4. The y_{test} contains the expected AUROCs for all remaining parameters. We choose the set of parameters that yields the highest expected AUROC, compute the AUROC, and append the results x_{train} and y_{train}. We iterate this process as many times as needed.

Figures 6-1 and 6-4 compare the convergence speed of a Gaussian process using 4 different kernels and a random search, for the Symlet-2 wavelet and the Gaussian-2

respectively. We see that the choice of the kernel is critical: the linear and absolute exponential kernels make the Gaussian process worse than the random search, while the squared exponential and cubic kernels perform significantly better than the random search.

The squared exponential kernel yields a slightly better result than the cubic kernel. In order to decide which one to use between the squared exponential kernel and the cubic kernel, we perform 100 searches for each kernel and look at the standard deviation of the AUROC as the number of computed AUROCs increases. The standard deviation is a very important property as in real conditions we will only perform one Gaussian process search, not 100. We therefore need a kernel that allows a reliable Gaussian process search, i.e. if we perform two searches the results should be similar (low standard deviation).

Figures 6-2 and 6-3 compare the standard deviation obtained with the squared exponential kernel and the cubic kernel for the Symlet-2 wavelet, Figures 6-5 and 6-9 show the same for the Gaussian-2 wavelet. We see that squared exponential kernel has a much smaller standard deviation than the cubic kernel.

The conclusion of this experiment is that the *squared exponential* kernel is the best choice as it obtains the highest solutions on average and has the smallest standard deviation.

Input: all_parameters_combinations, wavelet_name, number_of_experiments, number_of_initial_random_points, kernel

Output: x_{train}, y_{train}

```
1
2  // Variable initialization
3  x_train = x_test = y_train = y_test = []
4
5  // The first parameters are randomly chosen
6  for k = 1, k ≤ number_of_initial_random_points, k++ do
7  |  parameters = choose one set of parameter among all_parameters_combinations
8  |  all_parameters_combinations.remove(parameters)
9  |  x_train.append(parameters)
10 |  AUROC = compute_AUROC(parameter)
11 |  y_train.append(AUROC)
12 end
13
14 // The first parameters are chosen with a Gaussian process
15 experiments_number = number_of_initial_random_points
16 while experiments_number ≤ number_of_experiments do
17 |  x_test = all_parameters_combinations \ x_train
18 |  y_test = compute_expected_AUROCs(x_train, x_test, y_train)
19 |  best_parameters = parameter combination that gives the highest AUROC in
   |  y_test
20 |  output_file.write(max(y_test), best_parameters)
21 |  x_train.append(best_parameters)
22 |  y_train.append(compute_AUROC(best_parameters))
23 |  experiments_number = experiments_number + 1
24 end
25
26 return x_train, y_train
```

Algorithm 2: Gaussian process regression

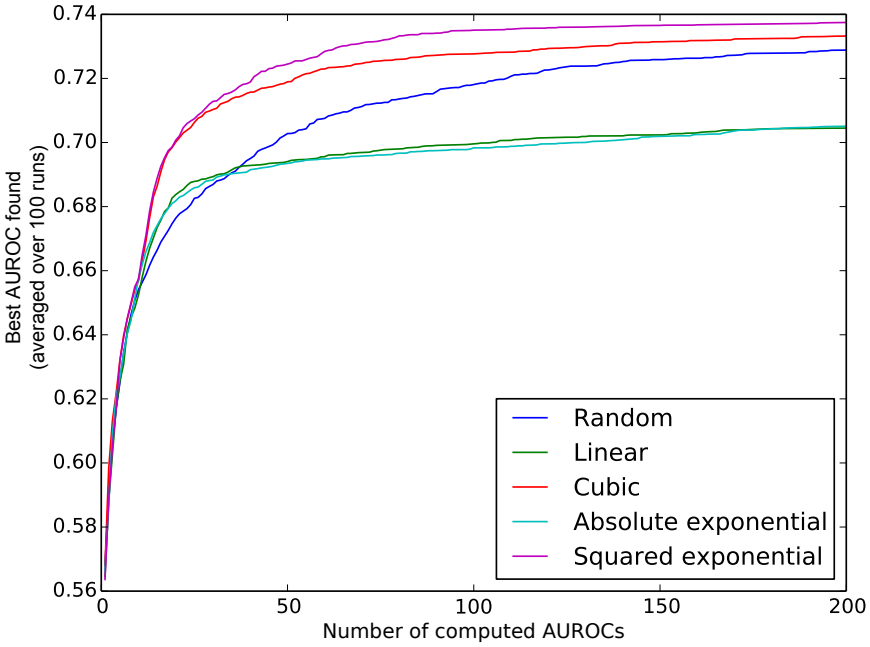

Figure 6-1: Impact of the kernel choice on the Gaussian Process with the Symlet-2 wavelet. The squared exponential kernel is the most optimal choice, closely followed by the cubic kernel. Both the squared exponential and the cubic kernel perform better than the random search. The linear and the absolute exponential kernels perform almost identically and are worse than the random search. Each plot was averaged over 100 runs.

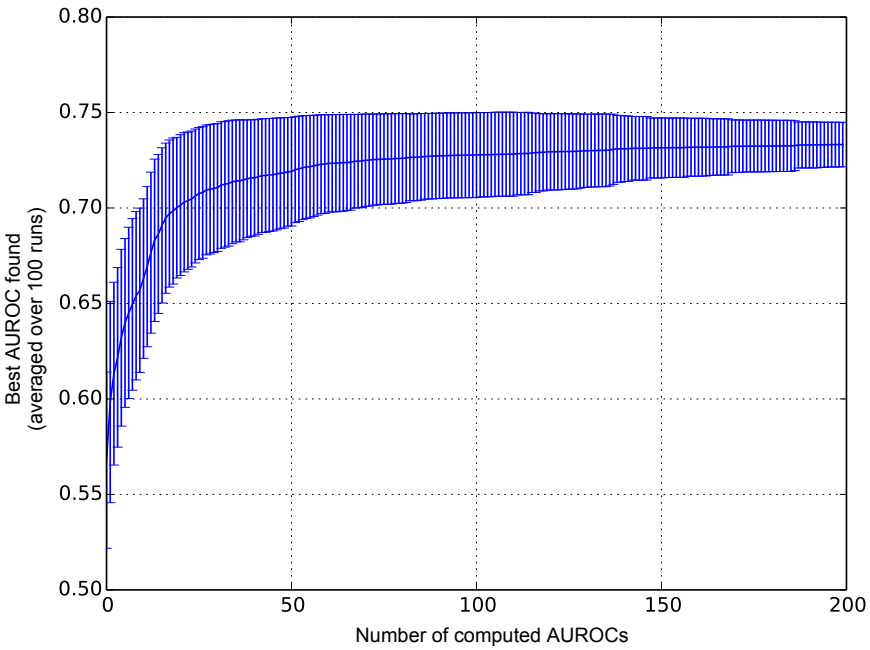

Figure 6-2: Standard deviation of cubic kernel with the Symlet-2 wavelet

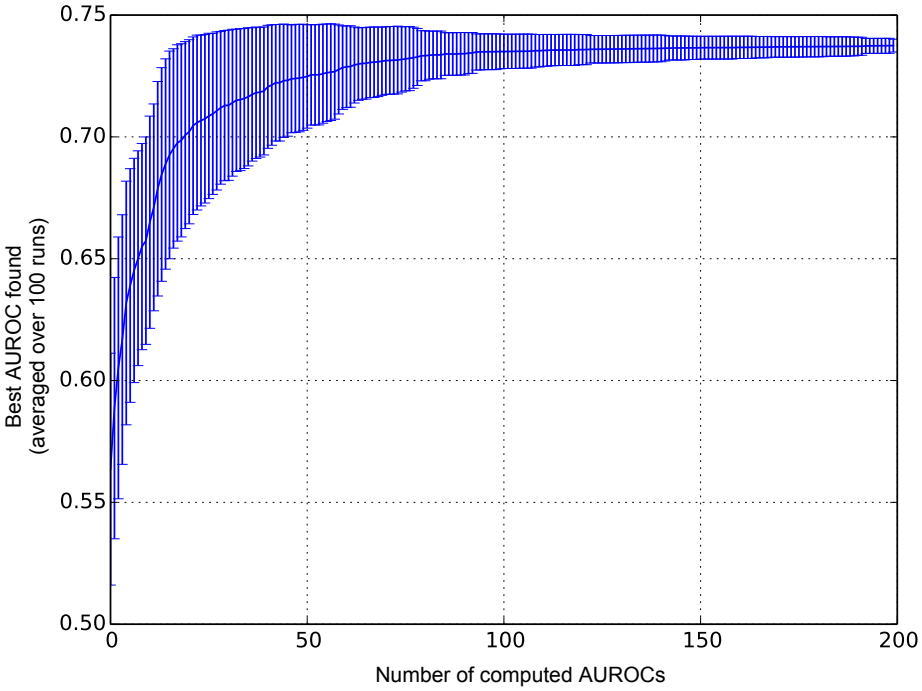

Figure 6-3: Standard deviation of the squared exponential kernel with the Symlet-2 wavelet

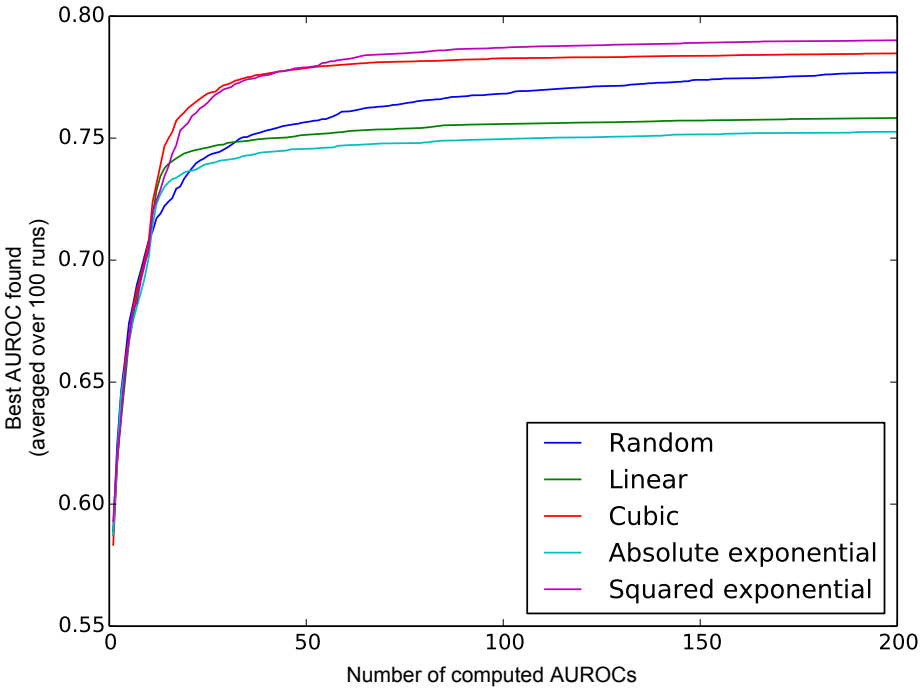

Figure 6-4: Same as Figure 6-1 but for the Gaussian-2 wavelet

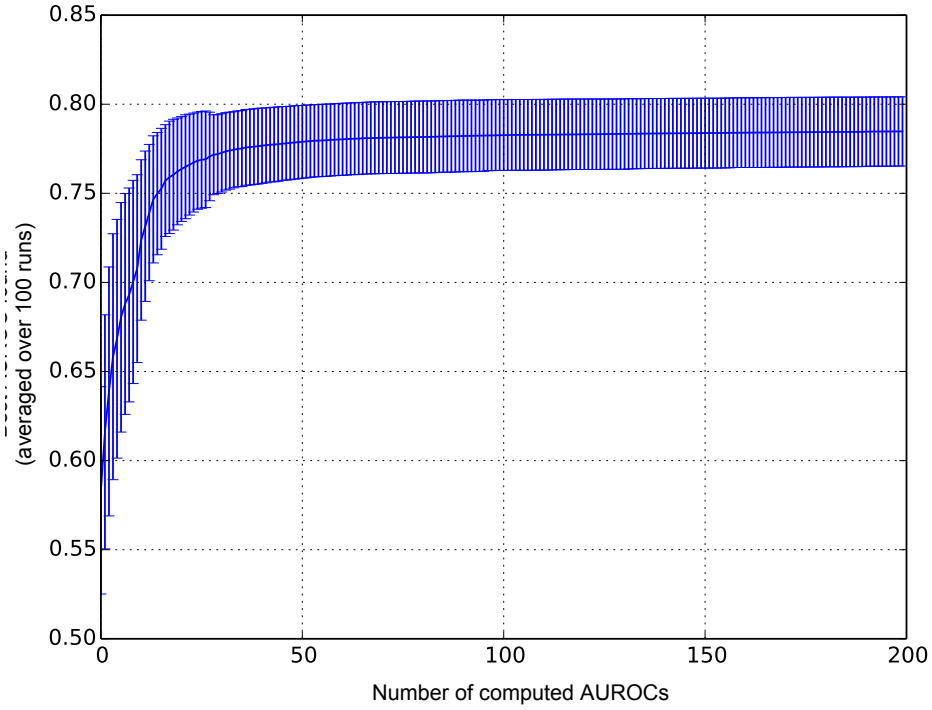

Figure 6-5: Standard deviation of the cubic kernel with the Gaussian-2 wavelet

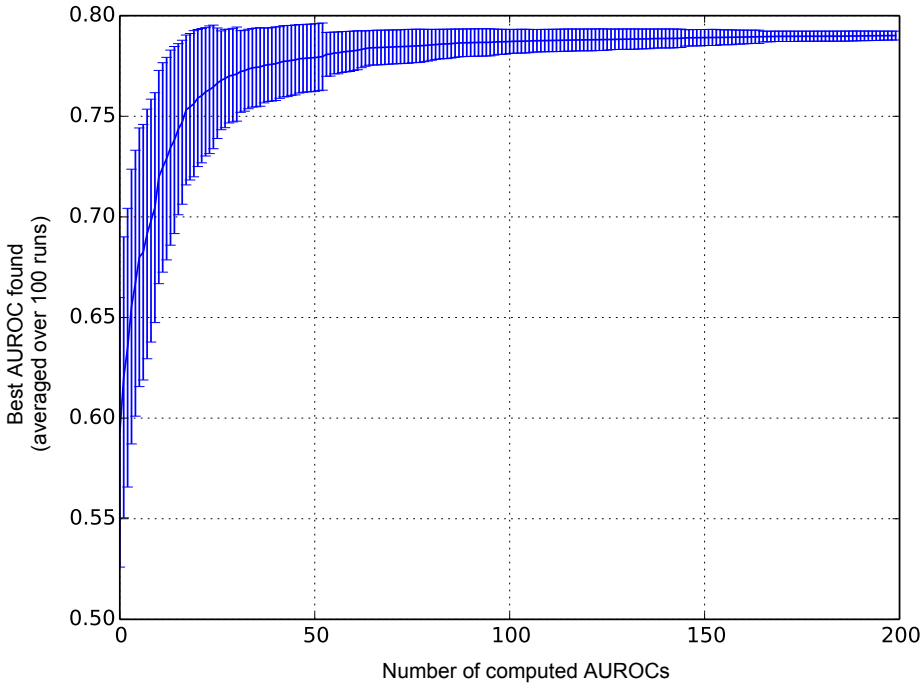

Figure 6-6: Standard deviation of the squared exponential kernel with the Gaussian-2 wavelet

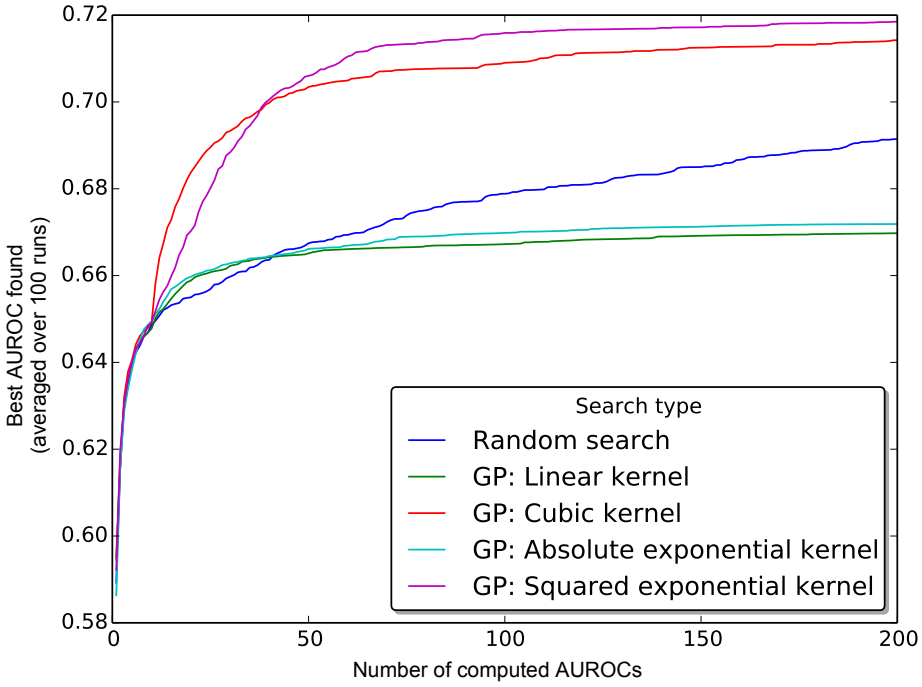

Figure 6-7: Same as Figure 6-1 but for the Haar wavelet

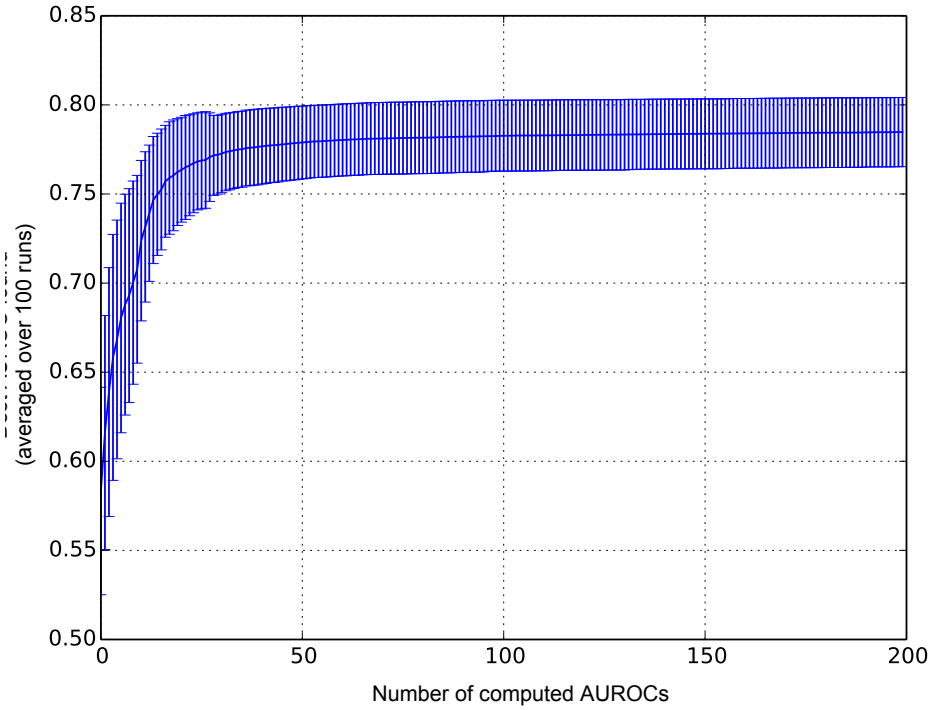

Figure 6-8: Standard deviation of the cubic kernel with the Gaussian-2 wavelet

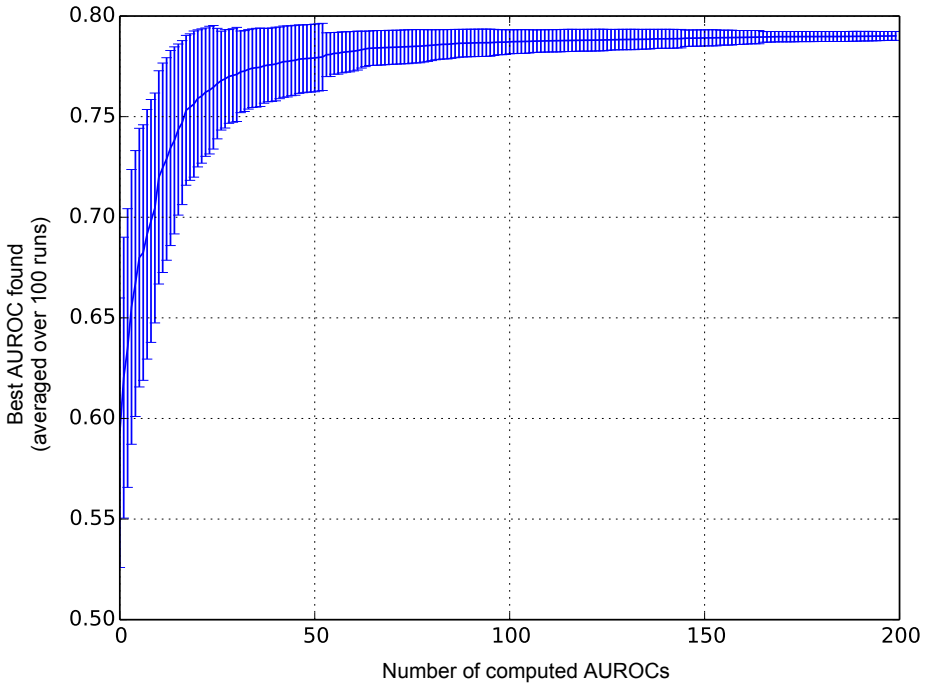

Figure 6-9: Standard deviation of the squared exponential kernel with the Gaussian-2 wavelet

6.2 Choosing the number of initial random experiments

Beyond the choice of the kernel, we also need to decide how many initial random experiments should be done before using the Gaussian process (i.e. the variable number_of_initial_random_points in Algorithm 2).

Using the squared exponential kernel we vary number_of_initial_random_points from 1 to 70 to see how it impacts the convergence speed. Figures 6-10, 6-11, and 6-12 present the results for the wavelets Symlet-2, Gaussian-2 and Haar respectively. The results show that computing 10 initial random experiments before using the Gaussian process is a good choice.

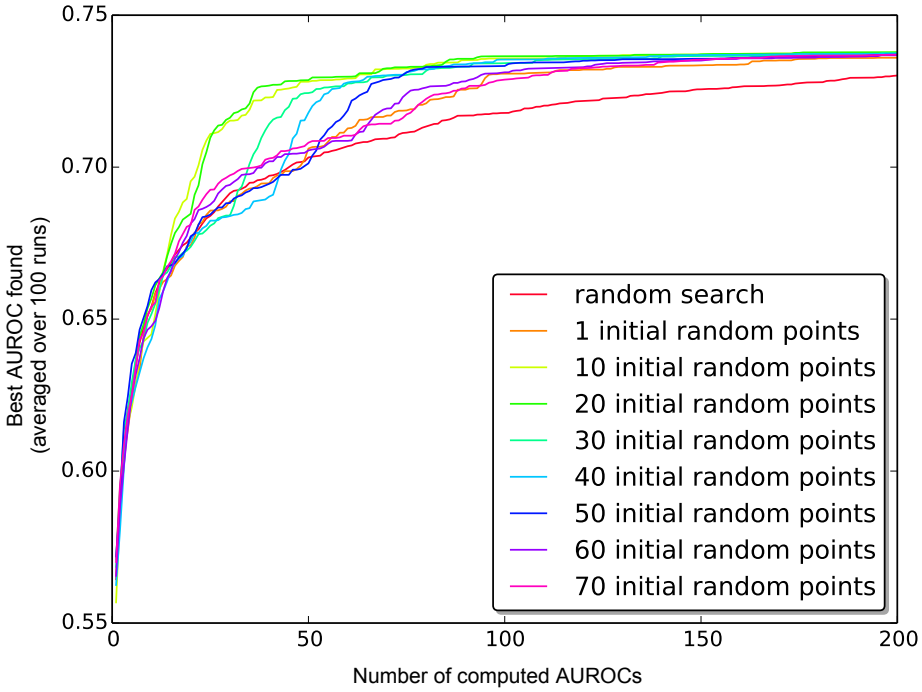

Figure 6-10: Impact of the choice of the number of random points on the search convergence speed, with the Symlet-2 wavelet. Choosing 10 random points is optimal.

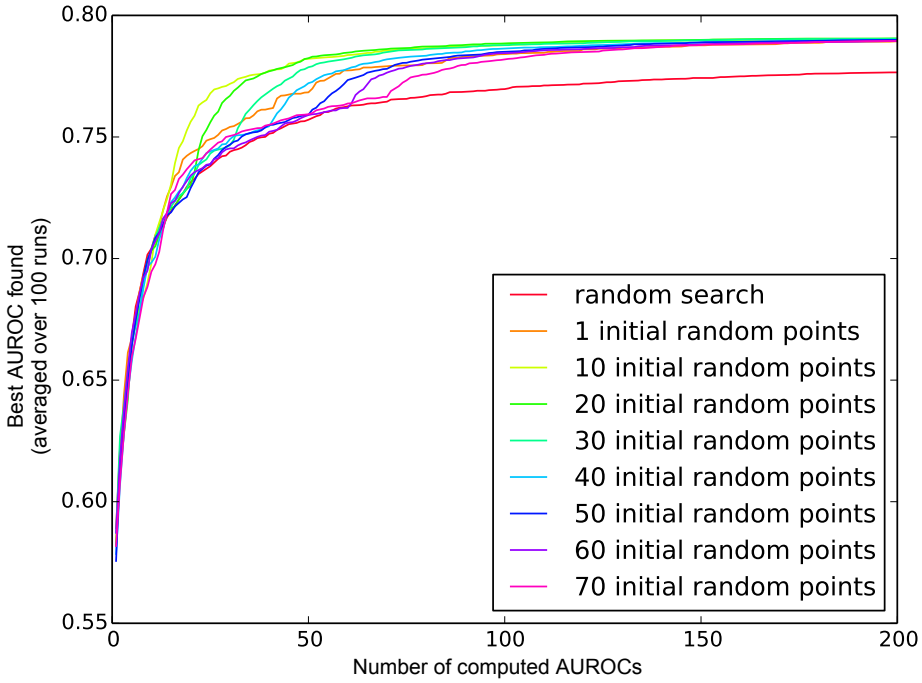

Figure 6-11: Impact of the choice of the number of random points on the search convergence speed, with the Gaussian-2 wavelet. Choosing 10 random points is optimal.

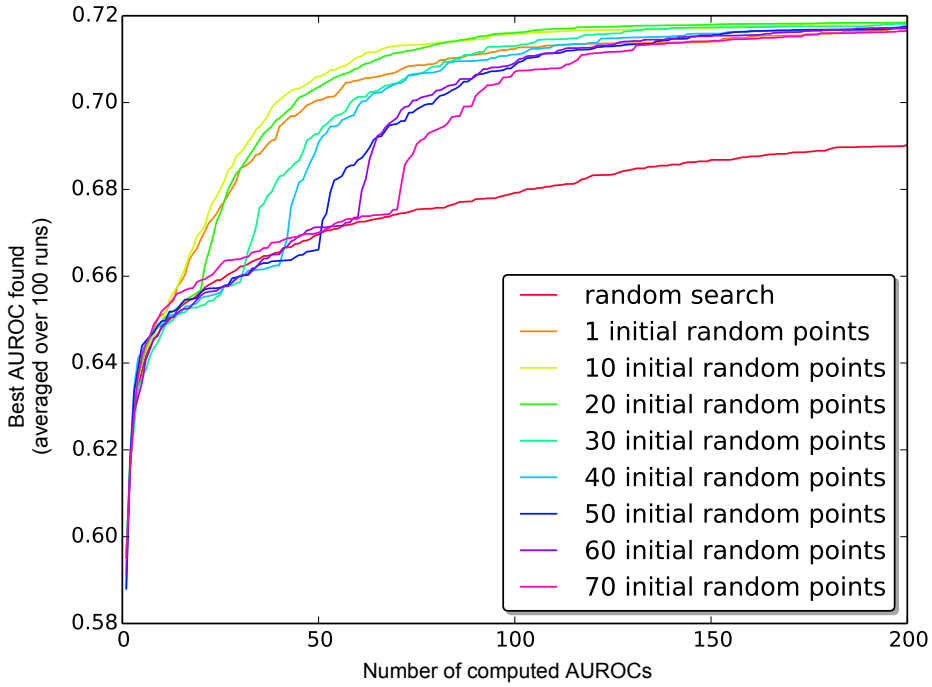

Figure 6-12: Impact of the choice of the number of random points on the search convergence speed, with the Haar wavelet. Choosing 10 random points is optimal.

6.3 Distributed Gaussian Process

In the previous experiments we have run the Gaussian Process on one machine only. In this experiment we analyze how the number of instances impacts the convergence speed of the Gaussian Process. Intuitively, increasing the number of instances should decrease the convergence speed because when one machine will fit its Gaussian Process to find the next parameter set to compute, it will not have access to the result of the tasks being computed by the other machines.

Figure 6-13 shows the results we obtain, which confirm our intuition.

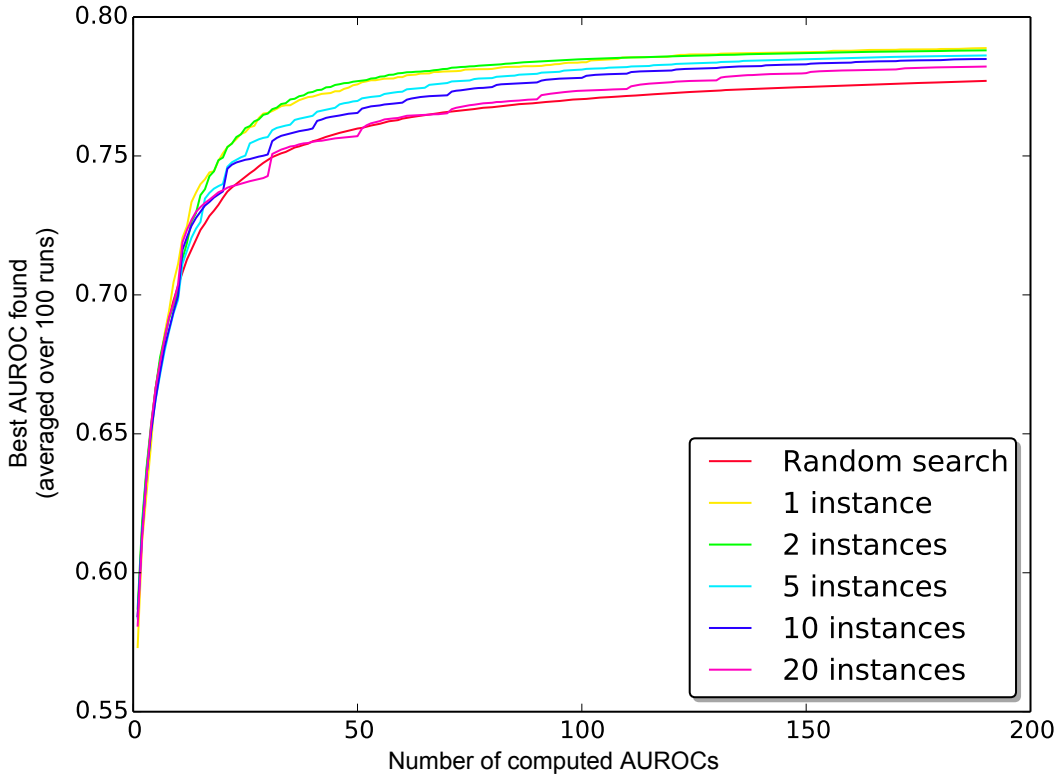

Figure 6-13: Distributed Gaussian Process: impact of the number of instances on the convergence speed, using the Gaussian-2 wavelet.

Chapter 7

Conclusions

7.1 Contributions

This thesis presented BeatDB. Our contributions are two-fold:

- From a methodological standpoint we designed a system both scalable and flexible. By analyzing existing studies on physiological signals in the literature, we pinpointed their commonalities and constructed a framework that can accommodate many of them. To achieve this goal, we elaborated a modular structure, each module having its own set of parameters. The modules handle beat detection, beat validation, feature extraction and event prediction. As a result, BeatDB allows to radically shrink the time an investigation takes since the latter corresponds to a certain parameterization of the system.

- From an experimental standpoint, as a real use case for BeatDB we presented many results regarding the features that can predict an AHE event. In particular, we showed wavelets can be used to predict an AHE event, but the choice of the wavelet transform, the scale as well as time shift is critical to obtain a high AUROC. Lastly, we demonstrated that a Gaussian Process help speed up the search for the right combination of parameters.

Throughout this work we showed the need for a system approach when mining knowledge from massive data sets. Beyond allowing faster investigations it is worth noting that taking a great care of the system design allows higher reproducibility. Prior to working on BeatDB, we worked on a similar project for education data to analyze student logs from Coursera and edX (Veeramachaneni et al., 2013; Dernoncourt et al., 2013a,b,d), which also combined faster investigations and higher reproducibility: we drew experience from this project when designing BeatDB.

BeatDB has already been used in published studies, such as Waldin (2013b) and Kim et al. (2014).

7.2 Future work

Building BeatDB required a lot of effort, in terms of design, coding and deployment. The numerous experiments with BeatDB allowed to thoroughly test the platform and investigate a real use case.

However, while the foundations of BeatDB are solid, many directions can be investigated to further enhance the platform:

- Patient clustering: as of now BeatDB takes patients randomly from the pool of all patients in the data set. It would be interesting to add a layer in BeatDB that would allow researchers to define conditions on which patients could be clustered. Patients have different medical backgrounds and different conditions, models that are trained on a specific group of patients should be able to perform better.

- More machine learning algorithms: it would be useful to extend BeatDB so that it can incorporate more machine learning algorithms. As most of the software is written in Python, scikit-learn (Pedregosa et al., 2011) is probably one of the best option. Since BeatDB is modular, the researcher can insert his own machine learning module, but offering more machine learning algorithms by

default would be more convenient.

- Feature selection: we choose features with low pair-wise correlation, but there exist more advanced feature selection methods, either filter-type, wrapper-type or embedded. Filter-type methods might be the best option as unlike wrapper-type methods they do not involve repeatedly invoking a learning algorithm and subsequently are much faster (Hall, 1999).

- Feature transformation: as an alternative to feature selection, we could reduce the dimensionality of data by using feature transformation, using for instance, principal component analysis or matrix factorization.

- Hyperparameter optimization: Gaussian Process was used to select the best scale and time shift parameters for the feature wavelets, but it might be interesting to tune other parameters and use other optimization algorithms. We could even combined hyperparameter optimization with machine learning algorithm selection (Thornton et al., 2013).

- From a technical standpoint, BeatDB can be deployed on an OpenStack cluster, but it could be useful to support other cloud computing platforms such as Amazon EC2. Also, given the amount of disk I/O, switching to an in-memory computing platform such as Hana (Färber et al., 2012a,b) should significantly boost the performances.

Beyond the platform itself, the experiments we carried out this thesis using the MIMIC database can be improved in many ways. Below are some ideas that we think would be worth investigating with BeatDB:

- As we have seen, the arterial blood pressure measurements are noisy. In practice, physicians and nurses in ICUs typically watch both the blood pressure and the ECG. Figure 7-1 shows both the ABP and the ECG: they are closely correlated, and since they are recorded using two different sensors we could use the ECG to read use the impact of the ABP noise on the predictions.

- Given that the definition of the AHE can change from physician to physician, it could be interesting to see how the prediction accuracy is influenced by the AHE definition that is used.

- So far we have only used the MIMIC Waveform database. However, a lot of structured information is contained in other part of the MIMIC database, namely the Clinical Database. There exists a mapping file[1] that links a subset of patients in the MIMIC Waveform database with their corresponding data in the MIMIC database, thereby enabling studies using both databases. Most studies only used the Clinical Database such as Ghassemi et al. (2014), some of them would tremendously benefit from capturing all the information present in the Waveform database in addition to the Clinical Database.

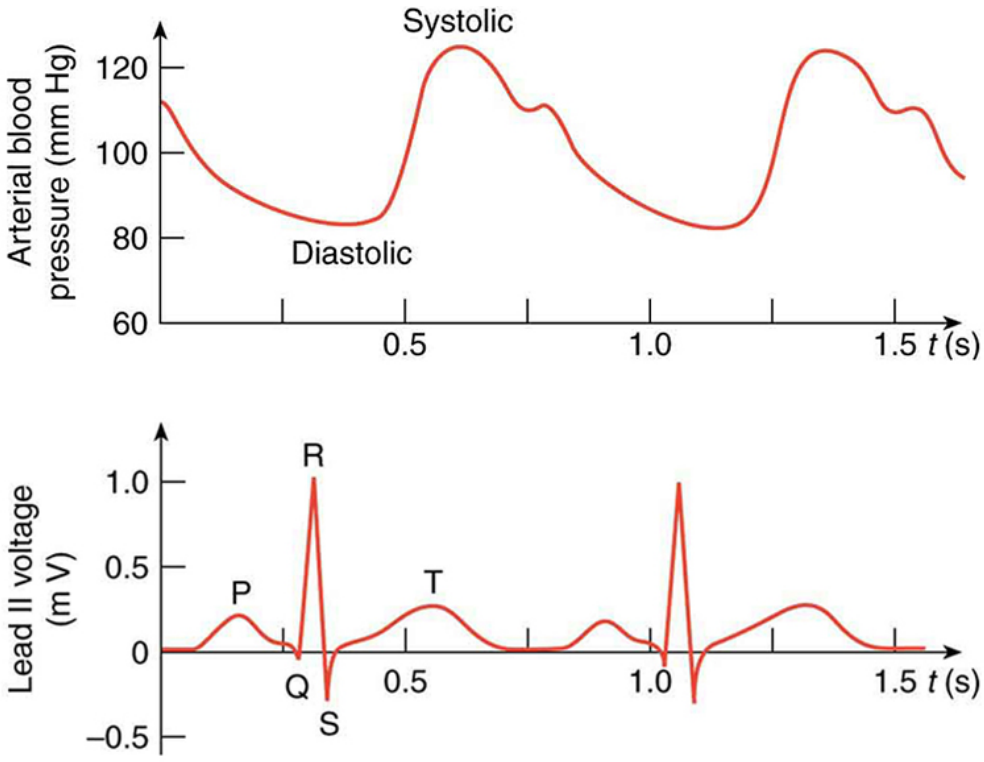

Figure 7-1: A lead II ECG with corresponding arterial blood pressure. Source: College (2012).

[1] http://physionet.org/physiobank/database/mimic2wdb/matched/

7.3 Conclusion

In this work we have built BeatDB, a large-scale machine learning and analytics framework, designed to make prediction studies on physiological signals significantly faster and easily reproducible. Even though a lot more work remain to be done, we hope to have laid a solid, fruitful ground for a profusion of extensions and future works, either from a system side to improve BeatDB or from an experimental perspective to use BeatDB to carry out more investigations.

Abbreviations

The following abbreviations are used in this thesis:

ABP	Arterial blood pressure
AHE	Acute hypotensive event
AUC	Area under the curve
AUROC	Area under the receiver operating characteristic curve
CWT	Continuous wavelet transforms
DBMS	Database management system
DWT	Discrete wavelet transforms
ECG	Electrocardiogram
GP	Gaussian process
ICU	Intensive care unit
MAP	Mean arterial pressure
MEWCP	Maximum edge weight clique problem
MIMIC	Multiparameter Intelligent Monitoring in Intensive Care
mmHg	Millimetres of mercury
RDBMS	Relational database management system
ROC	Receiver operating characteristic

Synonyms

The following terms are used inter-changeably in this thesis:

Central moment	Quadratic mean
Covariance function	Kernel
Crest factor	Square root of the peak-to-average ratio
False positive rate	Fall-out, hit rate, recall
Feature	Independent variable, covariate, explanatory variable
Intra-arterial catheter	Arterial line, A-line
Logistic regression	Logit regression
Loss function	Cost function
n^{th} moment about the mean	n^{th} central moment
Ordinary least squares	Linear least squares
Pressure transducer	Pressure sensor
Serialization	Marshalling, flattening, pickling
Stochastic process	Random process
True positive rate	Sensitivity

Bibliography

Abramowitz, M. and Stegun, I. A. (1972). *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*. Number 55. Courier Dover Publications.

Addison, P. S. (2005). Wavelet transforms and the ecg: a review. *Physiological measurement*, 26(5):R155.

Alted, F. and Fernández-Alonso, M. (2003). Pytables: processing and analyzing extremely large amounts of data in python. *PyCon 2003*.

Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 26–33. Association for Computational Linguistics.

Bell, R. M. and Koren, Y. (2007). Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79.

Bennett, J. and Lanning, S. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35.

Chandra, T., Ie, E., Goldman, K., Llinares, T. L., McFadden, J., Pereira, F., Redstone, J., Shaked, T., and Singer, Y. (2010). Sibyl: a system for large scale machine learning. *LADIS 2010*, 28.

College, O. (2012). Nerve Conduction and Electrocardiograms. http://cnx.org/content/m42352/latest/?collection=col11534/latest. [Online; accessed 01-July-2014].

Cornuéjols, A. (2005). Apprentissage et circulation d'information. *HDR, Université Paris-Sud, France*.

DBMS2 (2009). Greenplum is going hybrid columnar as well. http://www.dbms2.com/2009/10/14/greenplum-hybrid-columnar/. [Online; accessed 01-July-2014].

Dernoncourt, D., Hanczar, B., and Zucker, J.-D. (2014). Analysis of feature selection stability on high dimension and small sample data. *Computational Statistics & Data Analysis*, 71:681–693.

Dernoncourt, F. (2012). Replacing the computer mouse. *MIT CSAIL Student Workshop*.

Dernoncourt, F. (2014a). Age of empires is np-hard, even when playing alone.

Dernoncourt, F. (2014b). Trackmania is np-complete. *arXiv:1411.5765*.

Dernoncourt, F., Do, C., Halawa, S., O'Reilly, U.-M., Taylor, C., and Veeramachaneni, K. (2013a). Moocdb: Developing standards and systems for mooc data science. *MIT Technical Report*.

Dernoncourt, F., Taylor, C., O'Reilly, U.-M., Veeramachaneni, K., Wu, S., Do, C., and Halawa, S. (2013b). Moocviz: A large scale, open access, collaborative, data analytics platform for moocs. *NIPS 2013, Education Workshop*.

Dernoncourt, F., Veeramachaneni, K., and O'Reilly, U.-M. (2013c). beatdb: A large scalewaveform feature repository. In *NIPS 2013, Machine Learning for Clinical Data Analysis and Healthcare Workshop*.

Dernoncourt, F., Veeramachaneni, K., Taylor, C., and O'Reilly, U.-M. (2013d). Methods and tools for analysis of data from moocs: edx 6.002 x case study. Technical report, Technical Report, MIT.

Dobson, A. J. (2001). *An introduction to generalized linear models*. CRC press.

Drevo, W. (2014). Delphi: A distributed multi-algorithm, multi-user, self optimizing machine learning system.

Elkan, C. (2008). Log-linear models and conditional random fields. *Tutorial notes at CIKM*, 8.

Färber, F., Cha, S. K., Primsch, J., Bornhövd, C., Sigg, S., and Lehner, W. (2012a). Sap hana database: data management for modern business applications. *ACM Sigmod Record*, 40(4):45–51.

Färber, F., May, N., Lehner, W., Große, P., Müller, I., Rauhe, H., and Dees, J. (2012b). The sap hana database–an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33.

Fung, B. C., Wang, K., and Yu, P. S. (2007). Anonymizing classification data for privacy preservation. *Knowledge and Data Engineering, IEEE Transactions on*, 19(5):711–725.

Gad, H. (2008). Arterial blood pressure. http://faculty.ksu.edu.sa/dr.hayam/Lectures/Cardiovascular/Arterial%20blood%20pressure.pdf. [Online; accessed 01-July-2014].

Ghassemi, M. M., Richter, S. E., Eche, I. M., Chen, T. W., Danziger, J., and Celi, L. A. (2014). A data-driven approach to optimized medication dosing: a focus on heparin. *Intensive care medicine*, pages 1–8.

Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220.

Gomersall, C. (2014). Functions of monitoring. http://www.aic.cuhk.edu.hk/web8/haemodynamic%20monitoring%20intro.htm. [Online; accessed 01-July-2014].

Hall, M. A. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato.

Hamid Sheikhzadeh, R. L. B. A. S. C. (2007). . http://medicaldesign.com/components/making-better-sense-physiological-signals. [Online; accessed 01-July-2014].

Hellerstein, J. M., Ré, C., Schoppmann, F., Wang, D. Z., Fratkin, E., Gorajek, A., Ng, K. S., Welton, C., Feng, X., Li, K., et al. (2012). The madlib analytics library: or mad skills, the sql. *Proceedings of the VLDB Endowment*, 5(12):1700–1711.

Hemberg, E., Veeramachaneni, K., Dernoncourt, F., Wagy, M., and O'Reilly, U.-M. (2013a). Efficient training set use for blood pressure prediction in a large scale learning classifier system. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 1267–1274. ACM.

Hemberg, E., Veeramachaneni, K., Dernoncourt, F., Wagy, M., and O'Reilly, U.-M. (2013b). Imprecise selection and fitness approximation in a large-scale evolutionary rule based system for blood pressure prediction. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 153–154. ACM.

Hern, A. (2014). Google: 100,000 lives a year lost through fear of data-mining. http://www.theguardian.com/technology/2014/jun/26/google-healthcare-data-mining-larry-page. [Online; accessed 01-July-2014].

Jones, E., Oliphant, T., and Peterson, P. (2001). SciPy: Open source scientific tools for Python. http://www.scipy.org. [Online; accessed 01-July-2014].

Kennedy, H. L., Whitlock, J. A., Sprague, M. K., Kennedy, L. J., Buckingham, T. A., and Goldberg, R. J. (1985). Long-term follow-up of asymptomatic healthy subjects with frequent and complex ventricular ectopy. *New England Journal of Medicine*, 312(4):193–197.

Kim, Y. B., Seo, J., and O'Reilly, U.-M. (2014). Large-scale methodological comparison of acute hypotensive episode forecasting using mimic2 physiological waveforms. In *Proceedings of the 2014 IEEE 27th International Symposium on Computer-Based Medical Systems*, pages 319–324. IEEE Computer Society.

Koebler, J. (2014). Canadian insurance companies can legally practice genetic discrimination. http://bit.ly/CanadaGeneDiscrimination. [Online; accessed 18-July-2014].

Komorowski, M. (2009). A history of storage cost. http://www.mkomo.com/cost-per-gigabyte. [Online; accessed 01-July-2014].

Li, C., Zheng, C., and Tai, C. (1995). Detection of ecg characteristic points using wavelet transforms. *Biomedical Engineering, IEEE Transactions on*, 42(1):21–28.

Liu, M. and Zhang, D. (2014). Sparsity score: A novel graph-preserving feature selection method. *International Journal of Pattern Recognition and Artificial Intelligence*.

Macambira, E. M. (2002). An application of tabu search heuristic for the maximum edge-weighted subgraph problem. *Annals of Operations Research*, 117(1-4):175–190.

Macambira, E. M. and De Souza, C. C. (2000). The edge-weighted clique problem: valid inequalities, facets and polyhedral computations. *European Journal of Operational Research*, 123(2):346–371.

MacKay, D. (2006). Gaussian processes basics. In *Gaussian Processes in Practice Workshop, Bletchley Park, UK*.

Malin, B. and Sweeney, L. (2005). A secure protocol to distribute unlinkable health data. In *AMIA Annual Symposium Proceedings*, volume 2005, page 485. American Medical Informatics Association.

Mann, S. (1997). Smart clothing: The wearable computer and wearcam. *Personal Technologies*, 1(1):21–27.

McGhee, B. H. and Bridges, E. J. (2002). Monitoring arterial blood pressure: what you may not know. *Critical Care Nurse*, 22(2):60–79.

McKinney, W. (2010). Data structures for statistical computing in python. In *Proc. 9th Python Sci. Conf*, pages 51–56.

Miller, J. (2011). Mathematicalmonk's lectures on machine learning. YouTube.

Miller, R. (2014). CitusDB Releases An Open-Source PostgreSQL Tool That Promises Better Database Performance. http://tcrn.ch/1dR1QOw. [Online; accessed 01-July-2014].

Moody, G. B. and Mark, R. G. (1996). A database to support development and evaluation of intelligent intensive care monitoring. In *Computers in Cardiology, 1996*, pages 657–660. IEEE.

Moody, G. B., Mark, R. G., and Goldberger, A. L. (2001). Physionet: a web-based resource for the study of physiologic signals. *IEEE Eng Med Biol Mag*, 20(3):70–75.

Narayanan, A. and Shmatikov, V. (2009). De-anonymizing social networks. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 173–187. IEEE.

Neamatullah, I., Douglass, M. M., Li-wei, H. L., Reisner, A., Villarroel, M., Long, W. J., Szolovits, P., Moody, G. B., Mark, R. G., and Clifford, G. D. (2008). Automated de-identification of free-text medical records. *BMC medical informatics and decision making*, 8(1):32.

Nickson, C. (2014). Arterial line and Pressure Transducer. http://lifeinthefastlane.com/education/ccc/arterial-line/. [Online; accessed 01-July-2014].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830.

PhysiologyWeb (2011). Mean Arterial Pressure Calculator. http://www.physiologyweb.com/calculators/mean_arterial_pressure_calculator.html. [Online; accessed 01-July-2014].

PhysioNet (2014). What are PhysioBank-compatible (or WFDB-compatible) formats? http://physionet.nlm.nih.gov/faq.shtml#physiobank-formats. [Online; accessed 01-July-2014].

Prelcic, N. G., Márquez, O. W., and González, S. (1996). Uvi wave, the ultimate toolbox for wavelet transforms and filter banks. In *Proceedings of the Fourth Bayona Workshop on Intelligent Methods in Signal Processing and Communications, Bayona, Spain*, pages 224–227. Citeseer.

Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14(2):117–134.

Slipetskyy, R. (2011). Security issues in openstack. *Master's thesis, Norwegian University of Science and Technology*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959.

Srejic, U. and Wenker, O. C. (2003). A-line or intra-arterial catheters. *Internet Journal of Health*, 3(1).

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., et al. (2005). C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment.

Sun, J., Reisner, A., and Mark, R. (2006). A signal abnormality index for arterial blood pressure waveforms. In *Computers in Cardiology, 2006*, pages 13–16. IEEE.

Swan, M. (2013). The quantified self: Fundamental disruption in big data science and biological discovery. *Big Data*, 1(2):85–99.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM.

Tung, J., Eriksson, N., Kiefer, A., Macpherson, J., Naughton, B., Chowdry, A., Do, C., Hinds, D., Wojcicki, A., and Mountain, J. (2011). Characteristics of an online consumer genetic research cohort. *American Society of Human Genetics*.

Unser, M. and Aldroubi, A. (1996). A review of wavelets in biomedical applications. *Proceedings of the IEEE*, 84(4):626–638.

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Vaughan, D., Robinson, N., Lucas, N., and Arulkumaran, S. (2011). *Handbook of Obstetric High Dependency Care*. John Wiley & Sons.

Veeramachaneni, K., Dernoncourt, F., Taylor, C., Pardos, Z., and OâĂŹReilly, U.-M. (2013). Moocdb: Developing data standards for mooc data science. In *AIED 2013 Workshops Proceedings Volume*, page 17. Citeseer.

Waldin, A. (2013a). DCAP: A Distributed Computation Architecture In Python. http://byterial.blogspot.com/2013/02/dcap-distributed-computation.html. [Online; accessed 01-July-2014].

Waldin, A. (2013b). Learning blood pressure behavior from large blood pressure waveform repositories and building predictive models.

Williams, C. K. and Rasmussen, C. E. (2006). Gaussian processes for machine learning. *the MIT Press*, 2(3):4.

Zheng, L., Sun, Z., Li, J., Zhang, R., Zhang, X., Liu, S., Li, J., Xu, C., Hu, D., and Sun, Y. (2008). Pulse pressure and mean arterial pressure in relation to ischemic stroke among patients with uncontrolled hypertension in rural areas of china. *Stroke*, 39(7):1932–1937.

Appendix A

Reading CSV files in Python: a benchmark

A large quantity of BeatDB data is stored as CSV files. It is interesting to notice the time it takes to load a CSV file into an array can tremendously vary depending on the library that is used and the way the CSV file is stored. We present in this section a benchmark of Python library to read a CSV file. Since data analysis often relies on this type of storage, we believe it is of general interest to take a great care of this kind of operation as it might influence of lot the efficiency of the system.

The results of the benchmark are presented in Table A.1. We compare the following libraries:

- Python's native CSV module: introduced with Python 2.3 (*import csv*), it provides an easy way to read a CSV file line by line either to print or populate a Python list.

- NumPy (Van Der Walt et al., 2011): a widely used, open-source package for scientific computing with Python, one of the main building block of Scipy (Jones et al., 2001) and often refer as an open-source alternative to Matlab as it provides a fairly similar syntax.

- Pandas (McKinney, 2010): a more recent, open-source package that focuses on high-performance, easy-to-use data structures.

Python's native CSV module performs pretty well, while NumPy's *loadtxt()* is much slower, which is not surprising as the function provides many more features. Pandas' *read_ csv()* is impressively faster than the first two.

The main reason why loading a CSV file into an array is slow is that the parser has to go over the text file and convert it into an array structure, and possibly performing some casting at the same time. In order to avoid this parsing step, we can alternatively store the arrays not as text but binary files. In the Python jargon, this operation is called *pickling*, whereby a Python object is converted into a byte stream. Unpickling is the inverse operation, whereby a byte stream is converted back into an object. Pickling is also known as serialization (most common term outside the Python world), marshalling, or flattening.

As a result, we use NumPy's serialization function *save()* to convert our CSV files to binary files. As shown in the table A.1, NumPy's deserialization function *load()* is order of magnitude faster than all previous functions we tried to read CSV files. We note that using the combination of *tofile()* and *fromfile()* for data storage is even quicker, but the binary files generated are are not platform independent (e.g. no byte-order or data-type information is saved, while the binary files generated by *save()* are platform independent.

The source code of the benchmark is available online[1]. We lacked the time to investigate HDF5-based Python libraries such as PyTables (Alted and Fernández-Alonso, 2003) and h5py but they look promising.

[1] http://softwarerecs.stackexchange.com/a/7510/903

	Computer 1	Computer 2
OS	Windows 7 SP1 x64	Ubuntu 12.04 x64 LTE
Python	2.7.6 x64	2.7.3
NumPy	1.7.1	1.8.1
Pandas	0.13.1	0.14.0
Python's CSV reader	1.3293	1.788
Python's CSV reader list	1.0714	1.4965
Python's CSV reader float cast	2.6241	2.849
NumPy's loadtxt()	13.3207	7.498
Pandas's read_csv()	0.3638	0.3355
NumPy's fromfile()	0.0122	0.0105
NumPy's load()	0.0244	0.0135

Table A.1: Benchmark of different Python function to reading CSV files. Each function was used to read a CSV file containing 1 column with 750,000 lines and a CSV file containing 1 column with 750,000 lines. The reported values are expressed in seconds and is the sum of the duration to read both CSV files. We averaged the duration over 20 runs. We don't report standard deviations for the sake of readability as they were very small (less than 10% of the average)

Appendix B

On privacy and anonymization of personal data

It is worth noting that, while people are increasingly embracing data-recording technologies and services, many concerns over data privacy are raised. For instance, if health records of individuals were made public, health insurances could take advantage of that information to discriminate among their clients (as reported by Koebler (2014), insurance companies can legally practice genetic discrimination in Canada), banks might adjust their personal loan rates, employers may eliminate some candidates based on their medical conditions, and so on. However, on the other side, as exemplified by Hern (2014) who reports that the search firm's CEO and co-founder, Larry Page, estimates 100,000 lives could be saved next year if mining of healthcare data was acceptable, access to data is the key to improve people's lives.

The debate privacy vs. life quality is beyond the scope of this thesis but is good to keep in mind when making trade-offs during the anonymization phase of a data set, whether it pertains to healthcare records, educational performances, financial information and so on. It is indeed a trade-off as the more accurate a data set is, the more likely it is to contains interesting information, while overly anonymizing data set might remove crucial information. For example, the data set MIMIC, which

contains data on patients in the ICU and which we will use later on in this thesis, was anonymized by changing the dates the patients' arrival to the ICU. This removal of information could be deleterious for instance if we were trying to analyze what time of the year is the busiest for the medical personnel in ICUs.

A lot of work has been done to try to create some protocols to anonymize data sets, such as Malin and Sweeney (2005) for health data or Fung et al. (2007) for classification data, and complementarily a lot of work has been done to try to de-anonymize data sets such as Narayanan and Shmatikov (2009), who entered the $1-Million Netflix Prize contest (Bennett and Lanning, 2007; Bell and Koren, 2007) and successfully applied their de-anonymization techniques to identify Netflix data for a number of specific members.

Appendix C

Column-oriented vs. row-oriented database

In this section we discuss our choice to use flat files instead of using a relational database management system (RDBMS), which might sound surprising at first.

Figure C-1 shows how we could have stored BeatDB in a traditional RDBMS. The database is centered around records. Each record contains many signal samples for a given signal type. We detect beats based on the signal samples, which we mark as valid, invalid or jump. Then, we compute different features for each beat. Lastly, we scan for the presence of some medical condition, as we will see in the next chapter.

Instead, as we have previously seen, for each signal type, beatDB stores the samples in one file for each record, the beat validity data in one file for each record, and we store the feature values in one file for each record and for each feature.

The reason behind this segmentation is that all data queries are at record-level and feature-level. For ABP samples and beat validity, if we had used a traditional row-oriented RDBMS, we could have set a clustered index on the record ID in the beats and signal-samples tables. Figure C-2 shows the point of using a clustered index: when defining such an index, the physical data records on the disk follow the index sort order. This data organization allows to drastically reduce the number of disk

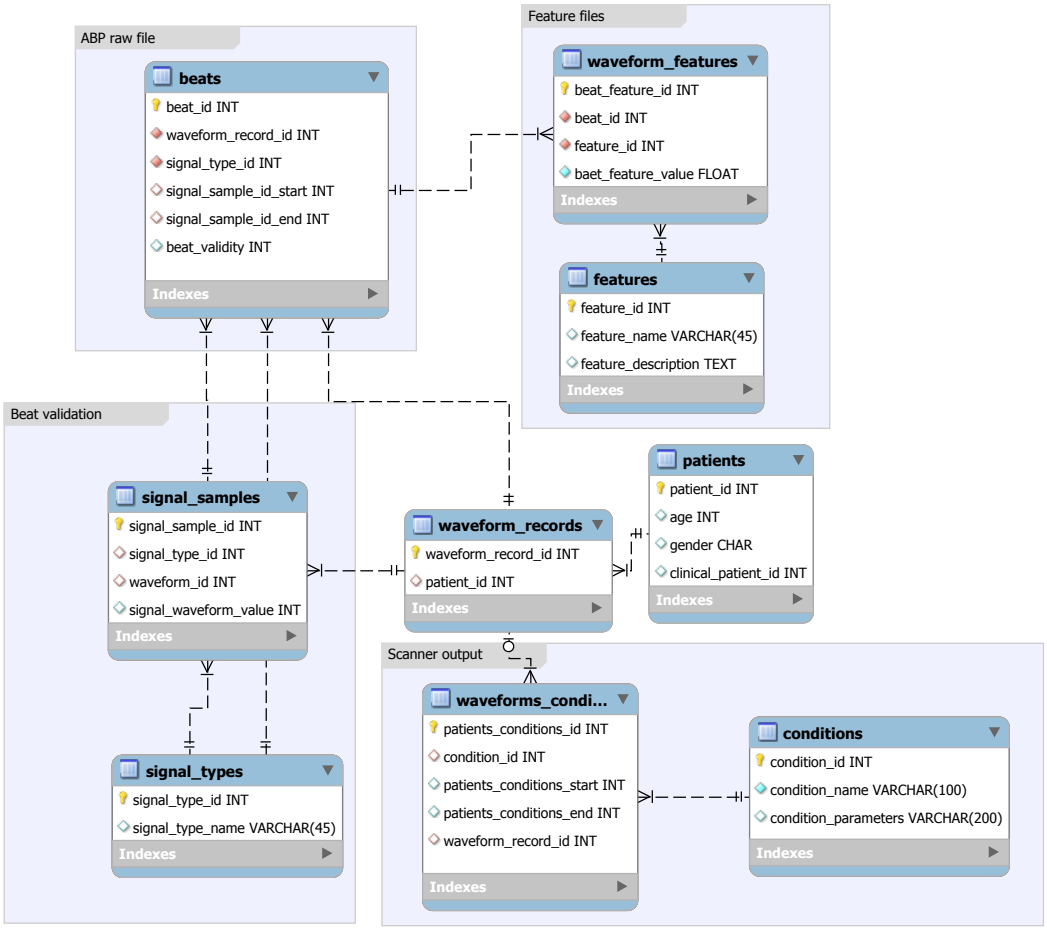

Figure C-1: RDBMS equivalent of the flat file design. Each layer indicates to which flat files each table corresponds to.

seeks when one value or a range of values of the clustered index needs to be retrieved. For example, if we set a clustered index on the record ID column in the beats table, retrieving all beats for one record would require only a few amount of disk seeks.

Setting a clustered index on the record ID in the beats and signal-samples tables would yield a read speed almost as fast as directly reading data from a flat file, but we did not have any use for any RDBMS feature so we stuck to flat files. Regarding the storage of the feature values, setting a clustered index in the RDBMS would not have been enough, as sometimes we want to retrieve a few features for all records, or all features for a few records, and there can only by one clustered index per table.

Our way of storing feature values in flat files corresponds to a column-oriented

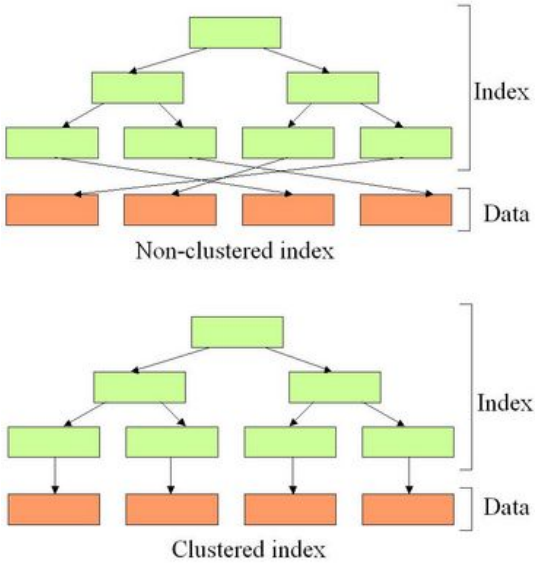

Figure C-2: Clustered vs. non-clustered index

database, since the columns (here the features) are stored in separate files. A column-oriented database stores the data of a table column by column on the disk, while a row-oriented database stores the data of a table row by row.

There are two main advantages of using a column-oriented database in comparison with a row-oriented database. The first advantage relates to the amount of data one's need to read in case we perform an operation on just a few features. Consider a simple query:

```
SELECT correlation(feature2, feature5)
FROM records
```

A traditional executor would read the entire table (i.e. all the features) as in Figure C-3. Instead, using our column-based approach we just have to read the columns which are interested in, as Figure C-4 illustrates.

The second advantage, which is also very important in our case since we have a large data set, is that column-based storage allows better compression, since the data in one specific column is indeed homogeneous than across all the columns.

The main drawback of a column-oriented approach is that manipulating (lookup,

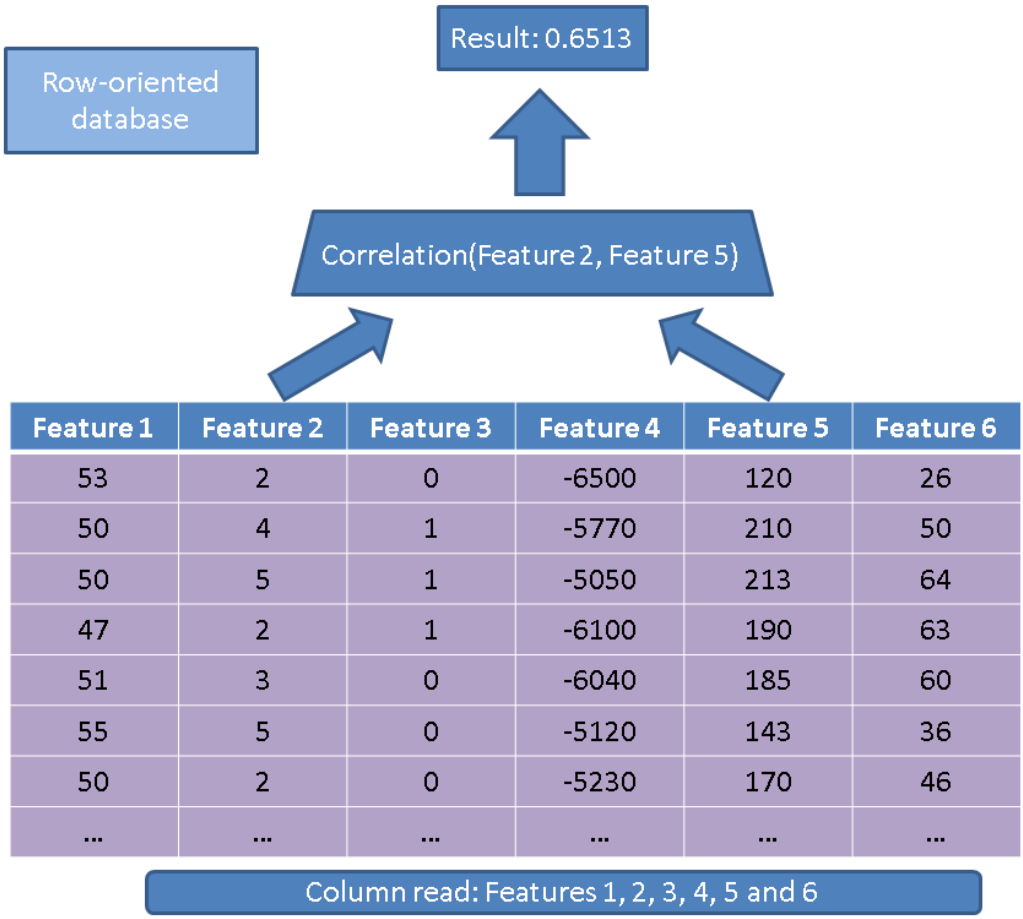

Figure C-3: Execution of the query when the data is stored in a row-based fashion. All columns must be read, because the data is stored row by row on the disk. An An alternative schema would be add the feature number as a column, and have the feature value as another column, as in Figure C-1 but this solution comes with its own drawbacks too.

update or delete) an entire given row is inefficient. However the situation should occur rarely in our case, since BeatDB is a database for analytics ("warehousing"), which means most operations are read-only, rarely read many attributes in the same table and writes are only appends.

Some RDMS offer a column-oriented storage engine option. For example, PostgreSQL has natively no option to store tables in a column-based fashion, but Greenplum has created a closed-source one (DBMS2, 2009). Interestingly, Greenplum is also behind the open-source library for scalable in-database analytics, MADlib (Hellerstein et al., 2012), which is no coincidence. More recently, CitusDB, a startup working on high-

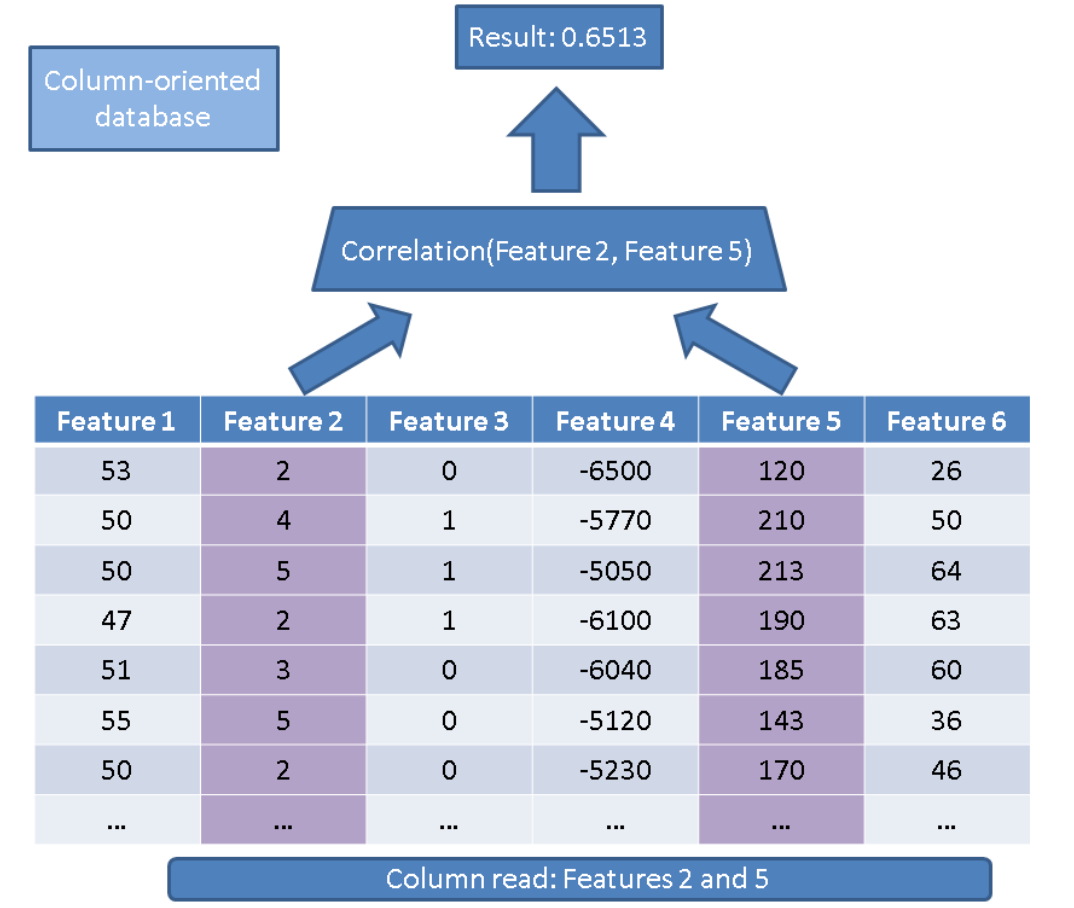

Figure C-4: Execution of the query when the data is stored in a column-based fashion.

speed, analytic database, released their own open-source columnar store extension for PostgreSQL, CSTORE (Miller, 2014). Google's system for large scale machine learning Sibyl also uses column-oriented data format (Chandra et al., 2010). This trend reflects the growing interest around column-oriented storage for large-scale analytics. Stonebraker et al. (2005) further discuss the advantages of column-oriented DBMS.

We chose eventually not to use an RDBMS with column-oriented storage for the sake of simplicity, and instead used flat files. Appendix A presents how we optimized reading from those flat files.

Appendix D

Machine learning techniques

D.1 Logistic regression

To predict the AHE events, we use logistic regression (aka. logit regression) which, despite its name, is a classification model, meaning that it is used to predict a some discrete variable (*classification*), unlike a regression model, which predicts a continuous variable (*regression*). The predicted variable is typically *categorical*, which means that its values (named classes or categories) have no intrinsic ordering, unlike ordinal variables, which do have an ordering. The predicted variable is also named dependent variable, output variable or target variable, and is commonly denoted y.

To make a prediction, the logistic regression take as input some features (aka. independent variables). The list of the feature values is denoted x. If we have d features, then $x \in \mathbb{R}^d$, assuming that features are real-valued, which is the case most of the time.

As logistic regression is a supervised algorithm, we fit the model using a *training set* that we denote $\left\{\left(x^{(i)}, y^{(i)}\right), i \in \{1, \ldots, n\}\right\}$, where each pair $\left(x^{(i)}, y^{(i)}\right)$ is the i$^{\text{th}}$ *training example*. The model we fit is the function $h : X \to Y$, where $h(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$. As we can see, the model is fully parameterized by θ, which is basically a weight vector that is applied to the feature vector through a linear combination. The

function g is the *logistic function* (hence the name logistic regression), which is a special case of the more general *sigmoid function*, even though the two are often confused.

By construction, y is between 0 and 1. We can interpret g in a probabilistic way: $P(y = 1) = h(x)$ and $P(y = 0) = 1 - h(x)$. The class affectation will be decided based on a threshold: for example, we can decide that if $P(y = 1) \geq 0.5$ then we consider that the class of y is 1. We will see the impact of the choice of the threshold in the next section.

In order to learn the model parameters θ using the training set, we must define a loss function (aka. cost function): $\text{Loss}(\theta) = \sum_{i=1}^{n} \left(\text{subcost}\left(h(x^{(i)}) - y^{(i)}\right)\right)$. Since we need to find the global minimum of the *Loss* function, we choose *subcost* in such a way that *Loss* is convex with respect to θ. If we use the mean squared error as the *subcost*, like in linear regression using ordinary least squares (aka. linear least squares), then *Loss* would not be convex (because function g is the logistic function). Instead, we chose $\text{subcost}(x, y) = y \log(h(x)) + (1 - y) \log(1 - h(x))$, which results in a *Loss* that is convex, and subsequently straightforward to minimize using gradient descent or some more advanced optimization algorithm such as conjugate gradient or BFGS (unlike linear regression, there is no closed-form expression for the optimal θ). This optimization can be interpreted probabilistically as a maximum log likelihood estimation because it finds $\arg\max_\theta (p(y|x))$.

Interestingly, logistic regression can be expressed in a more general framework: generalized linear model regression (Dobson, 2001). It can also be seen as a special case of a log-linear model (Elkan, 2008) and of a conditional random field.

D.2 Metrics

In order to assess the quality of our model, we choose the area under the curve of the receiver operating characteristic (AUROC), which we can use since the logistic

regression's outcome can be regarded as a probability as we have seen in the previous section.

Before presenting the ROC, the concept of confusion matrix must be understood. When we make a binary prediction, that can be 4 types of errors:

- We predict 0 while we should have the class is actually 0: this is called a true negative, i.e. we correctly predict that the class is negative (0). For example, an antivirus did not detect a harmless file as a virus .
- We predict 0 while we should have the class is actually 1: this is called a false negative, i.e. we incorrectly predict that the class is negative (0). For example, an antivirus failed to detect a virus.
- We predict 1 while we should have the class is actually 0: this is called a false positive, i.e. we incorrectly predict that the class is positive (1). For example, an antivirus considered a harmless file to be a virus.
- We predict 1 while we should have the class is actually 1: this is called a true positive, i.e. we correctly predict that the class is positive (1). For example, an antivirus rightfully detected a virus.

To be a confusion matrix counts, we go over all the predictions made by the model, and establish account for each of those 4 types of errors, as Table D.1 shows.

10 true positives (TP)	2 false negatives (FN)
3 false positives (FP)	35 true negatives (TN)

Table D.1: Example of a confusion matrix. Among the 50 data points that are classified in this example, 45 are correctly classified and the 5 are misclassified.

Since to compare two different models it is often more convenient to have a single metric rather than several ones, we compute two metrics from the confusion matrix, which we will later combine into one:

- False positive rate (FPR), aka. fall-out, hit rate and recall, which is defined as $\frac{FP}{FP+TN}$. Intuitively this metric corresponds to the proportion of negative data

points that are mistakenly considered as positive, with respect to all negative data points. In other words, the higher FPR, the more negative data points we will missclassified.

- True positive rate (TPR), aka. sensitivity, which is defined as $\frac{TP}{TP+FN}$. Intuitively this metric corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. In other words, the higher TPR, the fewer positive data points we will miss.

To combine the FPR and the TPR into one single metric, we first compute the two former metrics with many different threshold (for example $0.00; 0.01, 0.02, \ldots, 1.00$) for the logistic regression, then plot them on a single graph, with the FPR values on the abscissa and the TPR values on the ordinate. The resulting curve is called ROC curve, and the metric we consider is the AUC of this curve, which we call AUROC. Figure D-1 shows the AUROC graphically.

As the performance of the logistic regression is influenced by the choice of the training set, when assessing the performance of the logistic regression with a given set of features and other problem parameters, we will perform a 5-fold cross-validation and report the average AUROC obtained for each fold.

In 5-fold cross-validation we divide the data set into 5 chunks of equal size. We take 4 chunks as the training set, and we use the last chunk as the testing set to compute the AUROC. We perform this operation 5 times. This gives us 5 AUROCs, which we average to obtain a robust estimate of the logistic regression performance. Figure D-2 illustrates the process of averaging the 5 AUROCs.

It is worth noting that when dividing the data set in two different chunks we must ensure that the training data is independent from the testing data. One common mistake is to divide one patient's data into training and testing set, which violates this independence principle. For example if we split the blood pressure signal values of a patient, put the first half in the training data, and the second half in the testing data: since the second half partly depends on the first half, this breaks the independent

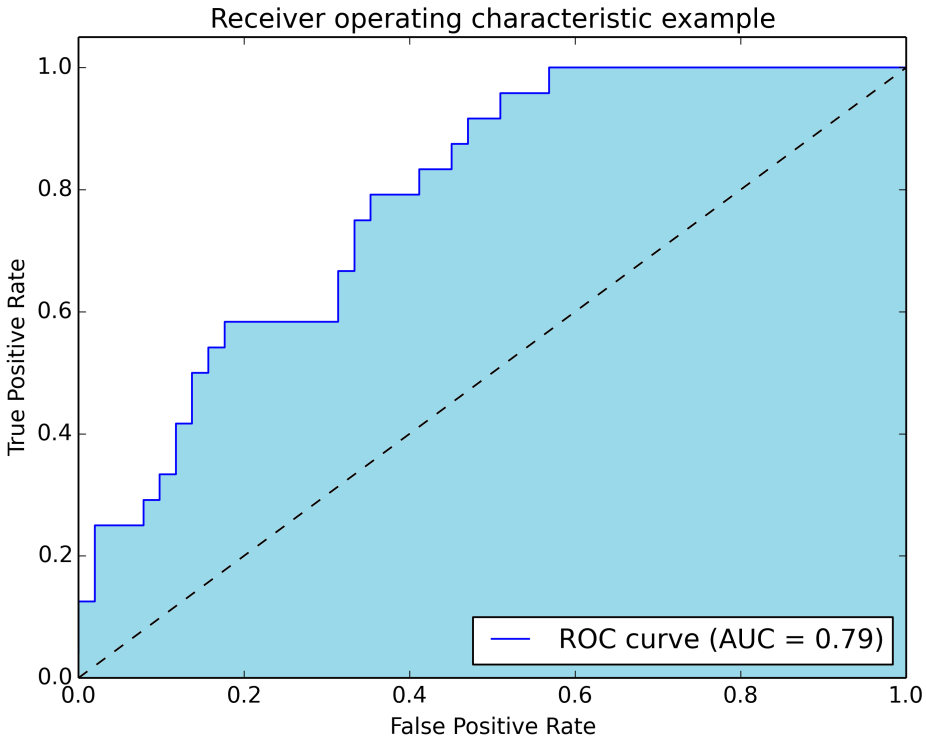

Figure D-1: Receiver operating characteristic (ROC) curve. The blue area corresponds to the Area Under the curve of the Receiver Operating Characteristic (AUROC). The dashed line in the diagonal we present the ROC curve of a random predictor: it has an AUROC of 0.5. The random predictor is commonly used as a baseline to see whether the model is useful.

condition. As a result, when dividing the data set into five different chunk, the data division is made on a patient-level basis.

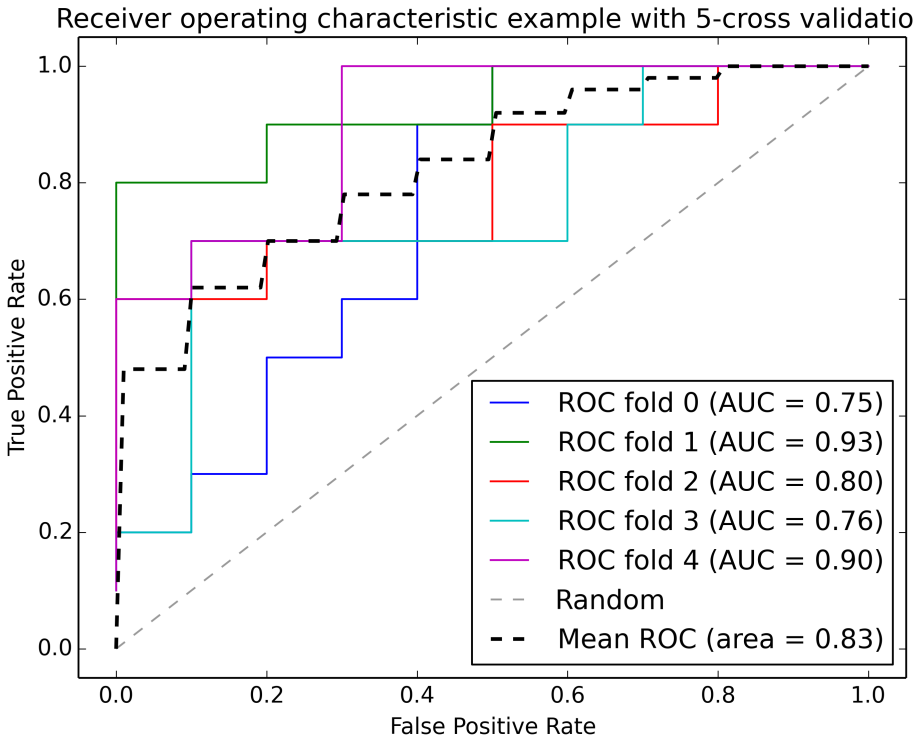

Figure D-2: ROC curves with 5-fold cross-validation: each cross-validation yields a ROC curve, hence the presence of 5 ROC curves. The grey dash line represents the random predictor, which we use as a baseline to see whether the model is useful as we have seen in Figure D-1. The bold dash line is the average of the 5 ROCs.

Appendix E

Gaussian process regression

This section was written based on Miller (2011) and Williams and Rasmussen (2006).

E.1 Gaussian process definition

The reference book on Gaussian Process for machine learning (Williams and Rasmussen, 2006) gives the following definition:

A Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalars or vectors (for multivariate distributions), a stochastic process governs the properties of functions. Leaving mathematical sophistication aside, one can loosely think of a function as a very long vector, each entry in the vector specifying the function value f(x) at a particular input x.

A Gaussian process is a particular case of a stochastic process (aka. random process, which simply designates a collection of random variables), which is often used to represent the evolution of some random value, or system, over time.

E.2 The mean vector and the variance-covariance matrix

A Gaussian distribution is completely specified by its mean (μ) and variance (σ^2). Its probability density function is:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

As a result, a Gaussian process is completely specified by its mean function and co-variance function. We can discretize it by saying that a discrete Gaussian process is completely specified by a mean vector and a co-variance matrix. In the case of the Gaussian process regression we'll focus on discrete Gaussian processes, where each Gaussian distribution represents an output in the training, validation or test set.

E.3 The intuition behind a covariance matrix

The covariance matrix expresses how tightly related random variables are between each other. As we can see in Figure E-1, which shows a probabilistic graphical model, the farther away random variables are from each other (e.g. y_1 and y_5 are far from each other), the lower is their covariance. The probabilistic graphical model shown in the figure also illustrates that the partial correlation in the inverse covariance matrix is equal to zero if and only if X is conditionally independent from Y given Z, with the assumption that all involved variables are multivariate Gaussian (the property does not hold in the general case).

E.4 The regression problem

The data is the following:

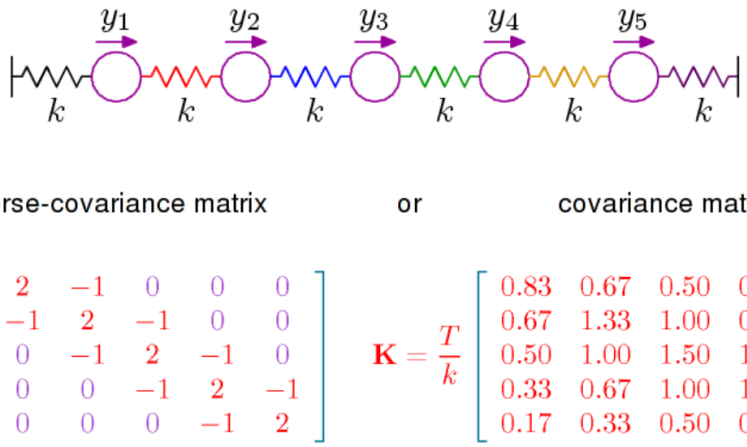

Figure E-1: Covariance matrix. The y_i are Gaussian random variables. Source: MacKay (2006)

- Training set inputs: $x_{train} = (x_1, x_2, ..., x_n)$
- Training set outputs: $y_{train} = (y_1, y_2, ..., y_n)$
- Test set inputs: $x_{test} = (x_{n+1}, x_{n+2}, ..., x_m)$

Note that:

- $x_i \in \mathbb{R}^D$ where D is the number of dimensions of the each input, that is to say the number of features each input has. This means that x_{train} is a matrix of dimension $D \times n$.
- $y_i \in \mathbb{R}$, i.e. the targets are real-valued. This means that y_{train} is a vector of n elements.
- We have n inputs in the training set and $(m - n + 1)$ in the test set.

The goal of the regression is to find $P(y_{test}|y_{train}, x_{train}, x_{test})$.

Since all random variables in X and Y are Gaussian, we have a closed-form expression for the distribution of the outputs that we were trying to find:

$$P(y_{test}|y_{train}, x_{train}, x_{test}) \sim \mathcal{N}(\mu_{y_{test}}, \sigma^2_{y_{test}})$$

where:

- $\mu_{y_{test}} = \mu_{test} + K_{test-train}(K_{train-train})^{-1}(y_{train} - \mu_{train})$
- $\sigma^2_{y_{test}} = K_{test-test} - K_{test-train}(K_{train-train})^{-1}K_{train-test}$

If we assume that the observed outputs y_{train} have some noise following a Gaussian distribution $\mathcal{N}(0, \sigma^2_{noise})$, then the two above formulas would be:

- $\mu_{y_{test}} = \mu_{test} + K_{test-train}(K_{train-train} + \sigma^2_{noise}I_n)^{-1}(y_{train} - \mu_{train})$
- $\sigma^2_{y_{test}} = (K_{test-test}+\sigma^2_{noise}I_{m-n+1})-K_{test-train}(K_{train-train}+\sigma^2_{noise}I_n)^{-1}K_{train-test}$

We have defined so far all the variables used in these formulas to compute the mean vector and the covariance matrix of $P(y_{test}|y_{train}, x_{train}, x_{test})$ except K. K is the covariance matrix of (x_{train}, x_{test}), and we shall see in the next section how to compute it.

E.5 Computing the covariance matrix

In the context of Gaussian process for prediction, the covariance matrix can be seen as the distance matrix between all the inputs (training and test sets put together). The choice of the covariance function, i.e. the function that we use to compute the covariance between two inputs, will deeply impact our predictions. The covariance function is typically denoted by k, as it is also called kernel.

We will investigate 4 different kernels:

- Linear kernel: $k(x, y) = x^T y$, the regression will be linear, as the kernel indicates.
- Cubic kernel: $k(x, y) = 3 \times \left(\left(x^T y\right)^2 + 2\left(x^T y\right)^3\right)$.
- Absolute exponential kernel (aka. Ornstein-Uhlenbeck): $k(x, y) = e^{|x-y|}$. The Ornstein-Uhlenbeck process is a stationary Gaussian process.

- Squared exponential kernel: $k(x, y) = e^{-0.5|x-y|^2}$, the regression will be non-linear (it can be shown to be mathematically equivalent to the Bayesian Linear Regression with an infinite number of basis functions).

The only constraint when building a kernel is that the resulting covariance matrix K must be positive definite.

$$K = \begin{array}{c} \\ x_{train} \\ \\ \\ x_{test} \\ \\ \end{array} \overbrace{\begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}}^{x_{train}} \overbrace{\begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}}^{x_{test}} \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) & k(x_1, x_{n+1}) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) & k(x_n, x_{n+1}) & \cdots & k(x_n, x_m) \\ k(x_{n+1}, x_1) & \cdots & k(x_{n+1}, x_n) & k(x_{n+1}, x_{n+1}) & \cdots & k(x_{n+1}, x_m) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_n) & k(x_m, x_{n+1}) & \cdots & k(x_m, x_m) \end{bmatrix}$$

$$K = \left(\begin{array}{c|c} K_{train-train} & K_{train-test} \\ \hline K_{test-train} & K_{test-test} \end{array}\right)$$

E.6 Computational complexity

As previously mentioned, we have a closed-form expression for the distribution of the outputs, which is a Gaussian distribution. The computational complexity of the Gaussian process regression solely lies in the calculation of the mean $\mu_{y_{test}}$ and the variance $\sigma^2_{y_{test}}$ of the Gaussian distribution. The formulas for each of those two values involve matrix operations, where matrix size is in the order of m, where as a reminder m is the sum of the number of points in the training and test sets.

The mean and variance formulas involve:

- matrix subtractions, the cost of which is $O(m^2)$ by subtracting element one by one;

- matrix inversions, the cost of which is typically $O(m^3)$ (e.g. using the Gauss-Jordan elimination) and can be speed-up using the Strassen algorithm ($O(m^{2.807})$), Coppersmith-Winograd algorithm ($O(m^{2.376})$) or Williams algorithm ($O(m^{2.373})$);
- matrix multiplications, the cost of which is $O(m^3)$ and can be speed-up using the same algorithm as for the matrix;

As a result, the computational complexity of a Gaussian process regression is $O(m^3)$. This limits the methods to the situations where the number of data points is around 10000 or fewer.

Appendix F

Wavelet library

F.1 Choice of library

Our requirements for the wavelet library are the ability to:

- compute a continuous wavelet transform;
- perform computation in an efficient way, i.e. the implementation should be reasonably fast;
- specify the scale as well as the time shift;
- offer many different wavelet transforms.

We investigated the following wavelet libraries:

- *PyWavelets* (Discrete Wavelet Transform in Python)[1]: as its name indicates, it only computes discrete wavelet transforms.
- *SciPy*'s scipy.signal.cwt function (Python)[2]: it only has 1 or 2 continuous wavelet transforms and the documentation is poor. See [3] for more details.

[1] https://pypi.python.org/pypi/PyWavelets/

[2] http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.cwt.html

[3] http://stackoverflow.com/q/23730383/395857

- Uvi Wave 300 (MATLAB) (Prelcic et al., 1996): it is not maintained anymore, the official website is no more available, the library does not run on new version of Matlab unless modified and it offers only a few continuous wavelet transforms.
- Wavelet Image Compression Construction Kit[4]: the last release was in 1997, and it focuses on image processing.
- Class Library for Wavelet Transforms on Images[5]: as its name indicates, it focuses on image processing.
- Wavelet1d[6]: it only computes discrete wavelet transforms.
- GNU Scientific Library (GSL)[7]: it only computes discrete wavelet transforms.
- Nwave[8]: it only computes discrete wavelet transforms.
- Matlab's Wavelet Toolbox[9]: it is a comprehensive toolbox that can compute both discrete and continuous wavelet transforms. It offers 87 different wavelet transforms[10] and a pretty fast implementation.

We eventually selected Matlab's Wavelet Toolbox.

F.2 Benchmark of library

Figure F-1 and F-2 present a benchmark of the running time of cwt(), which is the function to compute the continuous wavelet transform in the toolbox, depending on which transform we choose. It is interesting to see the impact of the choice of the wavelet transform impact the speed of cwt(). We run the benchmark with the -singleCompThread option when starting MATLAB to force it to use a single computational thread. cwt() is passed a 1,000,000-sample random signal and asked

[4] http://www.geoffdavis.net/dartmouth/wavelet/wavelet.html
[5] http://herbert.the-little-red-haired-girl.org/en/software/wavelet/
[6] https://code.google.com/p/wavelet1d/
[7] http://www.gnu.org/software/gsl/
[8] https://code.google.com/p/nwave/
[9] http://www.mathworks.com/products/wavelet/
[10] http://stackoverflow.com/a/24399951/395857

to compute scales 1 to 10. The CPU is an i7-3610QM, with Matlab R2014a running on Windows 7 SP1 x64 Ultimate. All durations are averaged over 10 runs.

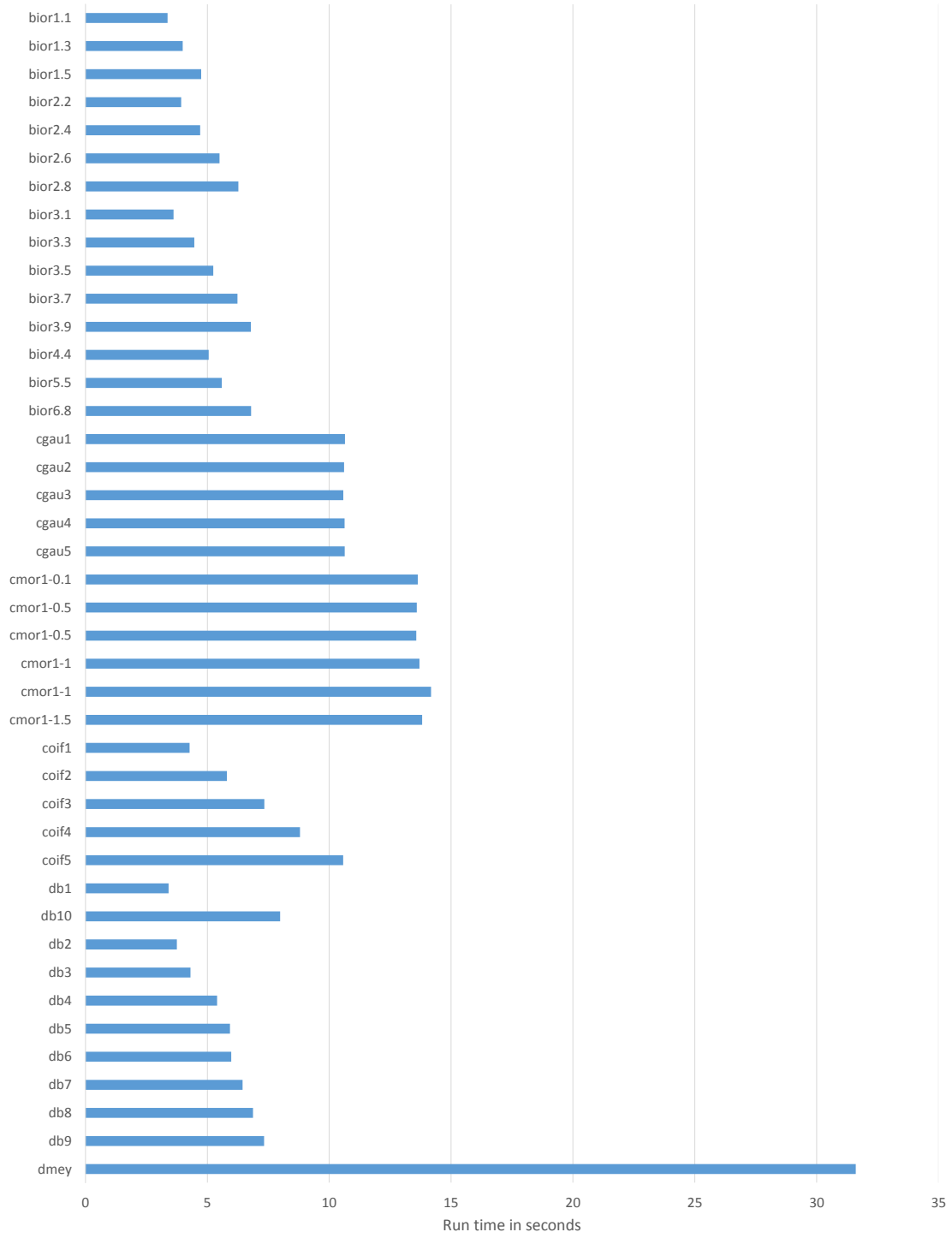

Figure F-1: Matlab's Wavelet Toolbox: cwt() benchmark. The choice of the wavelet transform has a high impact on the computation time.

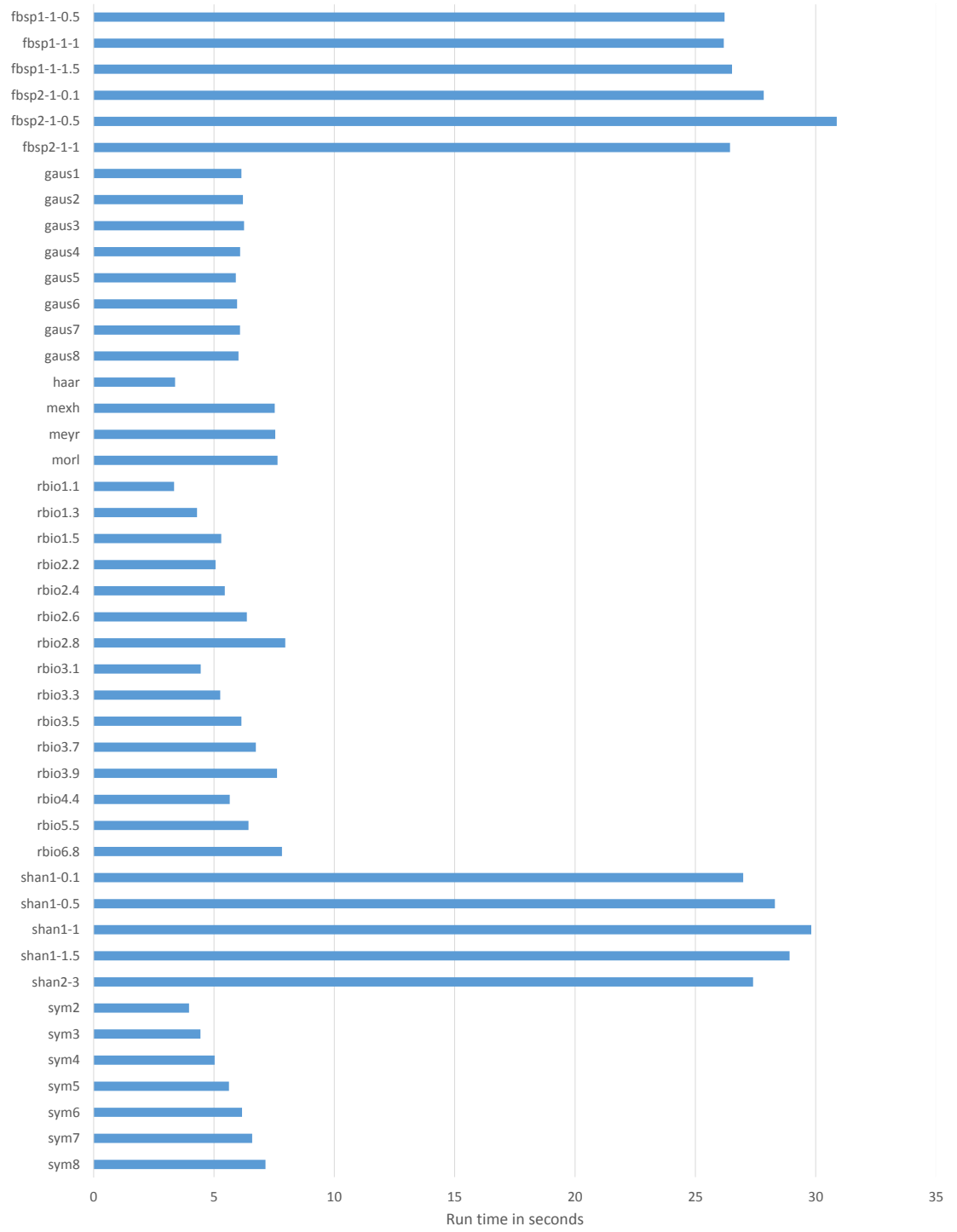

Figure F-2: Matlab's Wavelet Toolbox: cwt() benchmark. The choice of the wavelet transform has a high impact on the computation time.

Appendix G

Least correlated subset of wavelets from a correlation matrix

Due to time constraint we will not try all the wavelets as a features but just a few ones. In order to try to maximize the likelihood to find a good wavelet, i.e. a wavelet that when used as a feature help predict the AHE events, we want to select a subset of wavelets that is as least correlated as possible.

To put the problem more for formally let A be a correlation matrix, like the ones presented in Figures 5-4 and 5-5. In our case, the correlation matrix A is of dimension 65×65 since we have 65 different non-complex wavelets. A is the average of the correlation of the 65 wavelets across the first 10 scales on a 1,000,000-sample random signal.

Among these 65 wavelets, we would like to find a subset of wavelets, say 10 wavelets, whose correlation matrix contains as "little correlation" as possible. Quantifying how much "more correlation" a correlation matrix contains compared to another correlation matrix B requires to define a metric to quantify the level of correlation of a correlation matrix. One simple such metric is to take the mean of the absolute values of the non-diagonal elements of the correlation matrix, i.e. $\frac{2}{n^2-n}\sum_{1\leq i<j\leq n}|x_{i,j}|$ (we use the symmetry of the correlation matrix in this formula). With this metric the pro-

gram of finding the subset of wavelets with the least correlation becomes equivalent to the maximum edge weight clique problem (MEWCP).

A MEWCP is expressed as follows: we have an undirected weighted graph $G = (V; E)$ and we want to find the clique C of size at most n whose sum of the weights of the edges in C is the largest possible. As a reminder, a clique is a complete subgraph in a graph, i.e., subgraph where each pair of node of elements is connected by an edge. Figure G-1 shows an example of a clique. The MEWCP is NP-hard (Macambira and De Souza, 2000).

To see why our problem of finding the least correlated subset of wavelets of size n from a correlation matrix is equivalent to the MEWCP, let B be a matrix of the same dimension as A, such as $\forall(i, j), b_{i,j} = -|a_{i,j}|$. We consider B as the symmetric weighted adjacency matrix that expresses the undirected graph $G = (V; E)$ with weights associated to the edges in E corresponding to the values of B. Finding the clique C of size at most n whose sum of the weights of the edges in C is the largest possible yields the least correlated subset.

There exist some approximation algorithms for the MEWCP (Pullan, 2008; Macambira, 2002), which we can use to obtain a reasonably good subset of wavelets to use. Another way to make the problem easier would be to find another metric to quantify the level of correlation of a correlation matrix, such as the determinant[1] .

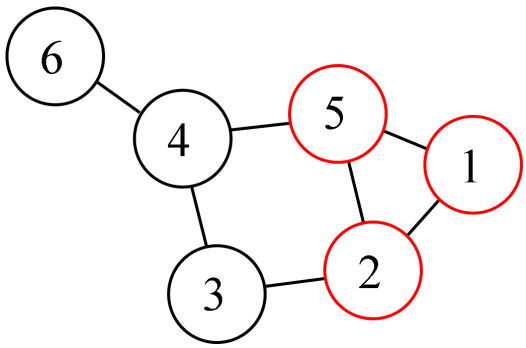

Figure G-1: Maximum clique in a graph

[1]To continue the discussion on the least correlated subset of random variables from a correlation matrix: http://stats.stackexchange.com/q/110426/12359

For the rest of this report, we choose 3 wavelets with low pair-wise correlation: Symlet-2, Gaussian-2 and Haar. There exist more advanced feature selection methods, which can be divided into filter-type, wrapper-type and embedded methods (Cornuéjols, 2005; Liu and Zhang, 2014; Dernoncourt et al., 2014), but we do not use them in this work.

For the rest of this thesis, we choose 3 wavelets with low pair-wise correlation: Symlet-2, Gaussian-2 and Haar. There exist more advanced feature selection methods, which can be divided into filter-type, wrapper-type and embedded methods, but we do not use them in this work.

Appendix H

Software design

The BeatDB code can be found on the GitHub repository: https://github.com/Franck-Dernoncourt/beatdb.

BeatDB is divided into three sets of tools:

- tools to populate BeatDB,
- tools to run workers,
- tools to analyze the results.

H.1 Populating BeatDB

H.1.1 Beat onset detection

The files mentioned in this section can be found in the folder /beat_detection.

The first step in populating BeatDB is to detect the beat onsets in the signal data of interest. We used WFDB[1] to detect the beat onset of MIMIC's ABP data. The latest version of WFDB, 10.5.17[2] at that time, had some issues with our Ubuntu

[1] http://www.physionet.org/physiotools/wfdb-linux-quick-start.shtml
[2] http://www.physionet.org/physiotools/archives/wfdb-10.5/wfdb-10.5.17.tar.gz

12.04 32-bit machine, so we used WFDB 10.5.9[3] instead.

We used the following step to install WFDB 10.5.9 on a Ubuntu 12.04 32-bit machine:

```
wget http://www.physionet.org/physiotools/archives/wfdb-10.5/wfdb-10.5.9.tar.gz
tar xfvz wfdb-10.5.9.tar.gz
cd wfdb-10.5.9
cd conf

emacs linux.def
edit at around line 30: (be careful to use this quote ‘, not ’):
LC = ‘curl-config --cflags‘
LL = ‘curl-config --libs‘
save the file

cd ..
sudo ./configure
sudo make install
sudo make check

# Linking
# you need to create a link libwfdb.so.10 that points to libwfdb.so:
sudo find / -iname libwfdb.so
cd /usr/lib
sudo ln -s /usr/lib64/libwfdb.so libwfdb.so.10
```

We do not need to compile wfdb-swig, we can actually use the libwfdbjava.so provided by wfdb-swig-matlab[4] .

Once WFDB is installed and libwfdbjava.so downloaded, we can run the Java program that extract the beat onsets:

[3] http://www.physionet.org/physiotools/archives/wfdb-10.5/wfdb-10.5.9.tar.gz
[4] http://www.physionet.org/physiotools/matlab/wfdb-swig-matlab/

```
javac -cp . abpbeatextraction/ABPBeatExtractionUnmatchedFull.java
java -classpath . abpbeatextraction.ABPBeatExtractionUnmatchedFull
javac -cp . abpbeatextraction/ABPextractMimic2v3.java
java -classpath . abpbeatextraction.ABPextractMimic2v3
```

H.1.2 Signal data transfer

The files in this section can be found in the folder /beat_transfer.

To retrieve the ABP data from MIMIC, we use the script Mimic2v3ABP.m, which can be run with the following command:

```
sudo apt-get -y install openjdk-6-jre openjdk-6-jdk
wget http://physionet.org/physiotools/matlab/wfdb-swig-matlab/install.sh
sudo bash install.sh -p http://physionet.org/physiotools/matlab/wfdb-swig-matlab
matlab -r Mimic2v3ABP
```

H.1.3 Beat validation

The files to perform section beat validation can be found in the folder /beat_validation. beat_validation.py contains all the rules to decide the validity of a beat.

H.1.4 Condition scanner

The files to perform section beat validation can be found in the folder /condition_scanner. The main file is condition_scanner.py.

H.1.5 Feature extraction

The first 14 features we use in this thesis were computed using the dcap framework. All the code can be found in the folder /feature_extraction.

The wavelet features were extracted using Matlab's Wavelet Toolbox. In order to make the extraction of wavelet more efficient we designed a specific code for it which can be found in the folder /feature_extraction/wavelet.

H.2 Worker logic

The worker logic is coded in /worker_logic/main_wrapper.py, which follows the algorithm described in Section 2.10. main_wrapper.py can be configured to do a grid search for a Gaussian process search.

OpenStack scripts to manage workers (adding new instances, cleaning broken instances, etc.) can be found in the folder /worker_logic/openstack. The interaction with OpenStack is done either by Eucalyptus euca2ools[5] or OpenStack command-line client nova[6]. Euca2ools can be used to interact with Amazon Web Services (AWS) as well, while nova is more specific to OpenStack. OpenStack indeed supports two APIS: the Amazon EC2 API (euca2ools) and the OpenStack API (novaclient).

H.3 Results analysis

The results analysis scripts are located in the folder /result_analysis. They output various graphs based on the results obtained by the worker logic. They can also simulate a Gaussian process search or a random search.

H.4 Benchmarks

The code for the CSV benchmark in Python can be found at http://softwarerecs.stackexchange.com/a/7510/903.

[5] https://github.com/eucalyptus/euca2ools
[6] http://docs.openstack.org/user-guide/content/install_clients.html

The code for the wavelet benchmark in the Matlab Wavelet Toolbox can be found at http://stackoverflow.com/a/24398613/395857.

H.5 Code statistics

The code was written in Python, Java, Matlab, Bourne Shell and SQL. Figure H-1 shows the programming language breakdown.

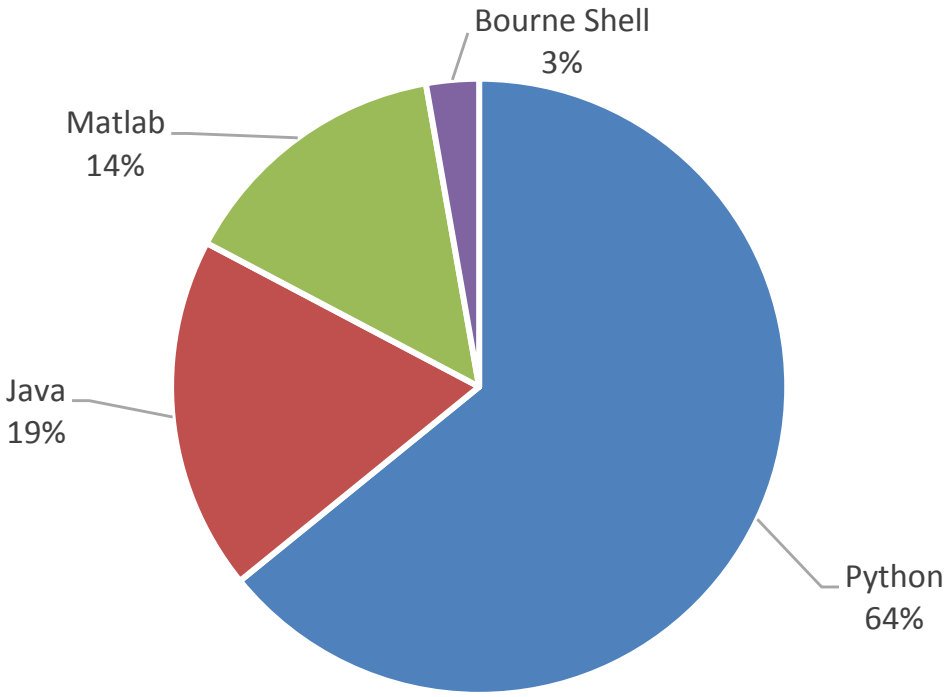

Figure H-1: Source lines of code (SLOC) per language. We used Python 2.7 and Matlab 2014a x64. In total there are around 10K SLOC (not counting comments and blank lines).