An R Package for Change Point Detection

Thales Mello

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Mestrado em Estatística

Orientador: Florencia Leonardi

São Paulo, 16 de Abril de 2019

An R Package for Change Point Detection

Esta é a versão original da dissertação elaborada pelo
candidato Thales Mello, tal como
submetida à Comissão Julgadora.

Acknowledgements

I thank Bruno M. Castro and Florencia Leonardi, for their work in the paper that inspired this project. I also thank specially Handley Wickham, for his work in several R packages and his tutorial on the development of R packages, and Yihui Xie for his work in the bookdown package. Without their contributions to the R community, none of this would be possible.

Resumo

Mello, Thales. **Um pacote R para detectar pontos de mudança**. 2019. 61 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2010.

Esta dissertação explora o desenvolvimento de um pacote da linguagem de programação R com o objetivo de detectar pontos de mudança em um conjunto de dados a ser analisado. Os pontos de mudança são estimados com base em uma função de verossimilhança de segmentos, a qual é escolhida de acordo com o tipo de problema que se deseja analisar. Os casos de uso abordados neste trabalho são (i) segmentos como blocos independentes de variáveis aleatória discretas e (ii) segmentos como blocos de variáveis aleatórias com mesmo parâmetro de regressão, sendo demonstrado um exemplo com regressão linear e outro com a média dos valores numéricos do segmento.

Neste trabalho são descritos os algorítmos de busca implementados pelo pacote (exato, hierárquico e misto), em que cada um possui características de performance e exatidão das estimativas próprios de cada algoritmo.

Por fim, são exploradas algumas aplicações de uso do pacote, bem como algumas tentativas de otimização de performance, explorando implementação de trechos críticos do código em linguagens de baixo nível, bem como se valendo também do uso de computação paralela.

Palavras-chave: segment, change-point, likelihood, R-package.

Abstract

Mello, Thales. **An R Package for Change Point Detection**. 2019. 61 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2010.

This dissertation explores the development of a package for the R programming language that allows the user to detect change points in a given data set. The change points are estimated using a segment likelihood function, which is chosen to suit the type of problem analyzed. The use cases mentioned in this work are (i) segments as blocks of independent discrete variables; and (ii) segments of random variables with a common regrassion parameter, for which a case of linear regression blocks and a case of Bernoulli distribution blocks are exemplified in this work.

This work also describes the search algorithms implemented by the package (exact, hierarchical and mixed), each one having characteristics of performance and quality of the results distinct from the others.

Finally, we explore some use-cases for the package, as well as some attempts at performance optimization, exploring the implementation of critical sections of code in low-level programming languages, as try to make use of parallel computing.

Keywords: segment, change-point, likelihood, R-package.

Contents

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**
1.1 Related Work 1

2 Main Definition **3**
2.1 A sequence of random variables and its segments 3
2.2 The segmentation problem 3
2.3 The Hausdorff distance 5

3 Solution Estimation **7**
3.1 Exact Algorithm 7
3.2 Hierarchical Algorithm 8
3.3 Hybrid algorithm 8

4 Simulations **9**
4.1 Segments of Independent Variables 9
4.2 Segments with similar averages 10

5 Real Data Examples **13**
5.1 Understanding the data 13
5.2 Building the likelihood 15
5.3 Penalizing the likelihood function 16
5.4 Reducing granularity 20
5.5 A non-usual likelihood function 21

6 Considerations on performance **27**
6.1 Benchmark the code 27
6.2 Native Code 27
6.3 Parallelization 28

7 Accuracy of algorithms **31**
7.1 Accuracy of estimates for the Berlin dataset 31
7.2 Accuracy of algorithms using different algorithms 31

8 Conclusion **35**

A Package Usage **37**

A.1 Installation 37

A.2 Usage 37

B Support Code **39**

Bibliography **53**

List of Figures

5.1 Average daily temperature over time as measured by different stations in Berlin . . . 14
5.2 Manual segmentation of the weather data picked according to intuition 14
5.3 Berlin weather data segmented using a non-penalized likelihood function 17
5.4 Example curve of the penalty function for $C_1 = C_2 = 1$, $s_1 = s_2 = 0.3$ and $L = 100$. 18
5.5 Berlin weather data segmented using an auto-penalized likelihood function 19
5.6 Berlin weather data segmented using an adjusted penalized likelihood function . . . 19
5.7 Average monthly temperature over time as measured by different stations in Berlin . 20
5.8 Downsampled Berlin monthly weather data segmented using an auto penalized likelihood function 21
5.9 Rescaled segments estimated for a downsampled monthly weather data of the Berlin data set using an auto penalized likelihood function 22
5.10 Berlin daily weather data segmented using an auto-penalized R-squared likelihood function 23
5.11 Berlin daily weather data segmented using an adjusted penalized R-squared likelihood function 23
5.12 Subset of Berlin daily weather data segmented using an auto-penalized R-squared likelihood function 25

List of Tables

4.1 Sample values of the correlated random variables defined in (4.1). 10
4.2 Segment tables as defined in (4.1). 10
4.3 Results of segmentation usign a non-penalized multivariate likelihood 11
4.4 Results of segmentation by aplying the likelihood function defined in (4.3) to a sample of size 10 of the model defined in (4.4). 12

5.1 First and last columns of the 'berlin' dataset 15
5.2 Expected values to be found when segmenting the Berlin dataset 15
5.3 Sample values of the linear regression likelihood function for different sizes of segment candidates 16
5.4 Results of the segmentation algorithm using a non-penalized likelihood function . . . 16
5.5 Results of the segmentation algorithm using an auto-penalized likelihood function . . 18
5.6 Results of the segmentation algorithm using an adjusted penalized likelihood function 20
5.7 Results of the segmentation algorithm using an auto-penalized likelihood function over downsampled berlin data 21
5.8 Results of the segmentation algorithm using an auto-penalized likelihood function over downsampled berlin data, but rescaled to fit the original data dimensions 22
5.9 Sample values of the R-squared likelihood function for different sizes of segment candidates 24
5.10 Results of the segmentation algorithm using an auto-penalized R-squared likelihood function over Berlin data 24
5.11 Results of the segmentation algorithm using an adjusted penalized R-squared likelihood function over Berlin data 25
5.12 Results of the segmentation algorithm using an auto-penalized R-squared likelihood function over a subset of Berlin data 25

6.1 Execution time comparison between native C++ and interpreted R code 28
6.2 Execution time comparison between parralel and single-threaded computation with a data set of 100 samples and 10 columns 28
6.3 Execution time comparison between parralel and single-threaded computation with a data set of 5 samples and 100 columns 29

7.1 Hausdorff distance for different segmentation attepmts over the Berlin weather data set 32

7.2 Comparison of solutions of different algorithms to segmenting the data set described in (4.1), measuring how far each solution is from the ideal solution using the Hausdorff distance. 32
7.3 Comparison of solutions of different algorithms to segmenting the data set described in (4.4), measuring how far each solution is from the ideal solution using the Hausdorff distance. 33

Chapter 1

Introduction

[Castro *et al.*, 2018] describe a method of segmenting a data set of a finite alphabet into blocks of independent variables. This work expands on that by generalizing the likelihood function used in the solution, showing it can be used for other use-cases, e.g. segments with homogeneous values (see Chapter 4.2), segments that have the same linear regression trend 5.

This work was developed as an R Programming Language [R Core Team, 2018] package. The packaging of the code into redistributable software was based on instructions provided by [Wickham, 2015], and the final code is made available as an R package [Mello e Leonardi, 2019]. This was made with the goal of making it as easy as possible to for researchers and R programmers to use this software. Source code for the package is also available as an open source software Mello [2019]. Finally, as a special implementation for the use case described by [Castro *et al.*, 2018] of segmenting finite alphabets, a specialized version of the multivariate likelihood function is implemented in native code using Rcpp [Eddelbuettel e François, 2011].

1.1 Related Work

[Maidstone *et al.*, 2017] also talks about optimally segmenting a data set, given a *cost* function to be minimized for the segments the data set is going to be split into. That is analogous to the segment likelihood used in this work, with the difference being that we look forward to maximize the likelihood function in this paper rather than minimize.

In their paper, they discuss search path algorithms different than the ones shown in this work. In their case, if the cost function satisfies certain conditions, the `fpop` and `snip` search algorithms can prune search path in a way that is mathematically optimal. They claim the algorithms presented in their work provide better results than previous work.

Notice that, because the approach used by [Maidstone *et al.*, 2017] is different than the one used by [Castro *et al.*, 2018], the `fpop` and `snip` algorithms couldn't be investigated and validated in time to be implemented in the Segmentr package. However, if in future works those algorithms provide good results estimation and performance compared to the currently implemented algorithms, they can be included in the Segmentr package. That will allow users to use the newly implemented algorithms with minimal code modification, by changing a single parameter.

Chapter 2

Main Definition

In this chapter we explain the problem by providing definitions of concepts and build equations that gradually help us explain the problem and make the case for our solution.

2.1 A sequence of random variables and its segments

Let X be a vector of random variables defined in (2.1), in which each X_j belong a set S. The equation also shows all random variables X_j have probability distribution F, each with parameter Θ_j, such that $X_j \sim F(\Theta_j)$. A segment is defined as a sequence of indices such that $\Theta_a = \Theta_{a+1} = \cdots = \Theta_b$ such that $1 \leq a \leq b \leq m$, in each a and b represent the first and last index of the segment, respectively. We also define a change point c as being an index in which there is a change in the probability distribution parameters, i.e. $\Theta_{c-1} \neq \Theta_c$, for $c \geq 2$.

$$\begin{gathered} X = (X_1, \ldots, X_m) \\ X_j \in S, \text{ for } 1 \leq j \leq m \text{ and } j \in \mathbb{Z} \\ X_j \sim F(\Theta_j) \end{gathered} \tag{2.1}$$

Putting it all together in (2.2), a data set X with t change points has each segment represented by the sequence $S_k = X_{c_k}, \ldots, X_{c_{k+1}-1}$, with c_k each representing a change point. In order to simplify equations, let $c_0 = 1$ and $c_{t+1} = m + 1$, even though they are not change points.

$$\begin{gathered} c_0 = 1 \\ c_{t+1} = m + 1 \\ 1 < c_1 < \cdots < c_k < \cdots < c_t < m + 1 \\ \text{, for } 0 \leq k \leq t \text{ and } k \in \mathbb{Z} \\ \Theta_{c_k} = \Theta_{c_k+1} = \cdots = \Theta_{c_{k+1}-1} \\ \Theta_{c_k-1} \neq \Theta_{c_k}, \text{ for } 1 \leq k \leq t \text{ and } k \in \mathbb{Z} \\ S_k = X_{c_k}, \ldots, X_{c_{k+1}-1} \end{gathered} \tag{2.2}$$

2.2 The segmentation problem

Let D be a $n \times m$ matrix defined in (2.3), with n rows and m columns. The matrix is comprised of n samples of the vector of random variables X, i.e. the column j in the matrix has values corre-

sponding to the random variable X_j. Let C, defined in (2.4), be the set of change points in the data set. Since we have that all random variables within a segment have the same probability function parameter, let L be the likelihood function whose maximizing argument, when given a segment S_k, is the probability distribution parameter Θ_k, as shown in (2.5). Therefore, the segmentation problem is defined as the process of estimating the set of change points C.

$$D = \begin{bmatrix} x_{11} & \dots & x_{1j} & \dots & x_{1m} \\ \dots & \dots & \dots & \dots & \dots \\ x_{i1} & \dots & x_{ij} & \dots & x_{im} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & \dots & x_{nj} & \dots & x_{nm} \end{bmatrix}_{n \times m} \tag{2.3}$$

$$C = \{c_1, \dots, c_t\} \tag{2.4}$$

$$\Theta_k = arg\,max\,\{L(\Theta|S_k)\} \tag{2.5}$$

One approach to find the change points set C is to maximize a likelihood function over the data set D as shown in (2.6). Considering the data set is compound by all the segments, and that each individual segment S_k will maximize the likelihood L for that portion of the data set, the likelihood of change points set C can be defined as the sum of the likelihoods of all the segments as shown in (2.7).

Notice that, in (2.6) and in (2.7), the segment likelihood function L and the data set X are parameters to the segmentation problem, which will ultimately be used to search for the set of change points C using dynamic programming. So, let s be a segmentation function such that $C = s(L, X)$, as defined in (2.8). Segmentr provides a few implementations of segmentation function that can be used to solve the segmentation problem.

$$C = arg\,max\,\{\mathcal{L}\,(C|X = D)\} \tag{2.6}$$

$$\mathcal{L}(C) = \sum_{k=0}^{t} L(S_k) = \sum_{k=0}^{t} L(\Theta|S_k) \tag{2.7}$$

$$C = s(L, X) = arg\,max\left\{\sum_{k=0}^{t} L(\Theta|S_k)\right\} \tag{2.8}$$

In summary, Segmentr is a generalization of the work described in [Castro *et al.*, 2018], in which the random variables of X are taken from finite alphabets and L is the multivariate estimate of discrete variables. In this work, besides analyzing the same problem described in [Castro *et al.*, 2018], the fact L is a parameter the segmentation function means it can be applied to many different scenarios in which X can manifest itself, e.g. segments with changes in the mean of columns, segments with different linear regression parameters. These cases are discussed in 4 and 5.

2.3 The Hausdorff distance

As the solution estimated by the segmentation function is a set of points, a measure is made necessary to compare the estimations provided by the different estimation algorithms that will be presented in Chapter 3. The Hausdorff distance is commonly used as a measure of distance between two distinct sets, and will be used to to compare change point sets with differing number of elements each.

[Munkres, 2000] defines the Haurdorff distance as measure of how far two subsets from a metric space are from one another. It's defined as the biggest of all the distances from a point in one set to the closest point in the other set, which is expressed mathematically by the equation defined in (2.9), in which X and Y represent sets given as inputs to the function, $d(x, y)$ represents a distance function defined in the metric space S such that $x, y \in S$.

$$d_H(X, Y) = \max \left\{ \sup_{x \in X} \inf_{y \in Y} d(x, y) \, , \sup_{y \in Y} \inf_{x \in X} d(x, y) \right\} \tag{2.9}$$

Since a set of change points is a subset of the column indices in a data set, we define the distance d as the absolute value of two given numbers for the use cases analyzed in this paper.

Chapter 3

Solution Estimation

In this chapter, we discuss the process to solve (2.8) and explore an solution involving dynamic programming. Because the exact solution to this problem has quadratic complexity, we also discuss some alternatives that simplify the search path and provide an approximate solution.

3.1 Exact Algorithm

The first attempt to try to solve the equation would be to actually search all the possible alternatives in order to find the correct minimal set of intervals that maximize the total likelihood of the system. It's done iteratively by:

- for each column index $i \in \{1, \ldots, p\}$
- for each column index $j \in \{i, \ldots, p\}$
- compute $l_{i:j} = \text{lik}(X_{i:j})$
- store the likelihood $l_{i:j}$ and the index j for the highest value of $l_{i:j}$ for later comparison

Finally, after computing all of the possible combinations of likelihood segments, the procedure to find the optimal solution would be as following:

- let $j = p$
- while $j > 1$, repeat the steps below
- search the stored values for $i = \arg\max_i(l_{i:j})$
- store i in the set of change points C if $i \neq 1$
- let $j = i$ and repeat until $j = 1$

Given the indices in the change points set C, let $c_0 = 1$, $c_{k+1} = p$ and $c_i \in C$ be the sequence of sorted elements of C, for $i = 1, \ldots, k$. The minimal set of intervals would be defined as $S_0 = \{c_{i-1} : c_i, \text{ for } i = 1, \ldots, k+1\}$ in which $c_{i-1} : c_i = \{\text{contiguous sequence of number from } c_{i-1} \text{ to } c_i\}$.

The described algorithm will provide the correct answer for the equation, precisely because it analyses all the possible combinations. However, that has a time complexity in Big-O notation of $O(np^2)$ in which n is the number of samples in the data set and p is the number of columns. Because a lot of use cases for data segmentation has usually a large number of columns and not many data samples, e.g. DNA data, computation time can be very prohibitive.

3.2 Hierarchical Algorithm

In order to try to simplify the search path of the algorithm, [Castro *et al.*, 2018] also proposes a technique that relies on assuming the data to be hierarchical, i.e. it assumes the minimal solution set can me estimated by only searching the data for the optimal change point segmentation once. The algorithm is described below.

- let d be the current data set we want to segment
- for each column i in the data set
- compute the total likelihood $l_i = \text{lik}(X_{1:i-1}) + \text{lik}(X_{i:p})$ if $i \neq 1$
- find i for which l_i is maximum
- if $l_i < \text{lik}(X_{1:p})$ return empty set as result of current function call
- recursively find the set of change points C_L by calling the algorithm on the left segment $X_{1:i-1}$
- recursively find the set of change points C_R by calling the algorithm on the right segment $X_{i:p}$
- return $C = C_L \cup C_R \cup \{i\}$ as result of current function call

Implementing the algorithm described above will allow for a time complexity of $O(np\log(p))$ where n is the number of samples, p is the number of columns. The reduction in time complexity is only possible because of the strong assumption the data set has a hierarchical likelihood behavior. However, that doesn't hold true for many situations, as it will be seen in more detail in the next chapters, and the use of this algorithm should done with care.

3.3 Hybrid algorithm

The hybrid algorithm is a modified version of the hierarchical algorithm, in which it will start searching the combinations using the hierarchical approach, and if the segments become smaller than a certain threshold, it will try to find the exact segments with the exact algorithm. The procedure would be as follow:

- let d be the current data set we want to segment
- if the number of columns of d is $p_d < k$, in which k is a predefined threshold, return the set of change points calculated by the exact algorithm.
- for each column i in the data set
- compute the total likelihood $l_i = \text{lik}(X_{1:i-1}) + \text{lik}(X_{i:p})$ if $i \neq 1$
- find i for which l_i is maximum
- if $l_i < \text{lik}(X_{1:p})$ return empty set as result of current function call
- recursively find the set of change points C_L by calling the algorithm on the left segment $X_{1:i-1}$
- recursively find the set of change points C_R by calling the algorithm on the right segment $X_{i:p}$
- return $C = C_L \cup C_R \cup \{i\}$ as result of current function call

The only difference of this procedure and the hierarchical procedure is the presence of the conditional step in the beginning of the procedure that tests whether of not to use the hybrid algorithm.

Chapter 4

Simulations

In order to exemplify the utility of this package, a handful of hypothetical data will be presented, together with a proposal of an appropriate likelihood function that is expected to segment the data in an expected manner.

4.1 Segments of Independent Variables

In order to show compatibility with the work presented in [Castro *et al.*, 2018], we analyze the same problem presented in their work.

Let $C_1, C_2, \ldots, C_n$ a sequence of n vectors of data points, each with m observations represented by $C_k = \{c_{k1},\ c_{k2},\ ...,\ c_{km}\}$, with $1 \leq k \leq n$. Assume that each C_k is a function of an unknown set of independent random variables X, such that $C_k = f_k(X)$. When used together with a multivariate likelihood [Park, 2017], for which an implementation is provided in the `segmentr` package as the `multivariate()` function, it's possible to identify the set of N_b change points $b_1, ..., b_{N_b}$, with $b_0 = 1$ and $b_{N_b+1} = n + 1$, such that all f_k, for $b_i \leq k \leq b_{i+1} - 1$ is a function of the same set of independent random variables X, or each $1 \leq i \leq N_b$.

Let $X_1, ..., X_6$ be independent random variables and $C_1, ..., C_{15}$ be defined as in (4.1) . It's possible to see the first segment $C_1, ..., C_5$ depend on X_1, X_2, the second segment $C_6, ..., C_{10}$ depend on X_3, X_4 and the third and last segment $C_{11}, ..., C_{15}$ depend on X_4, X_5. Therefore, the change points would be $b_1 = 6$ and $b_2 = 11$.

Table 4.1: *Sample values of the correlated random variables defined in* (4.1).

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
2	1	1	3	2	1	-1	2	3	1	1	-1	2	3	1
1	0	1	2	1	2	0	2	4	2	1	-1	2	3	1
2	0	2	4	2	2	0	2	4	2	2	0	2	4	2
2	0	2	4	2	1	-1	2	3	1	1	0	1	2	1
2	0	2	4	2	2	0	2	4	2	1	-1	2	3	1
2	1	1	3	2	2	0	2	4	2	2	0	2	4	2

Table 4.2: *Segment tables as defined in* (4.1).

Segment no.	First index	Last index
1	1	5
2	6	10
3	11	15

$$
\begin{aligned}
C_1 &= X_1 \\
C_2 &= X_1 - X_2 \\
C_3 &= X_2 \\
C_4 &= X_1 + X_2 \\
C_5 &= X_1 \\
C_6 &= X_3 \\
C_7 &= X_3 - X_4 \\
C_8 &= X_4 \\
C_9 &= X_3 + X_4 \\
C_{10} &= X_3 \\
C_{11} &= X_5 \\
C_{12} &= X_5 - X_6 \\
C_{13} &= X_6 \\
C_{14} &= X_5 + X_6 \\
C_{15} &= X_5
\end{aligned}
\tag{4.1}
$$

To illustrate, consider the following code, which defines $X_1, ..., X_6$ and $C_1, ..., C_{15}$ as specified in (4.1), and also consider the matrix D as the column binding of the vectors $C_1, ..., C_{15}$.

It's possible to identify the segments in the matrix D with a penalized version of the `multivariate` function included in the package, as shown in Table 4.2

Notice it's important to use a penalized likelihood version of the `multivariate` function because the original function favors bigger segments. The answer provided by the `segment` package in that scenario would actually be all the columns in the D matrix, as shown in Table 4.3.

4.2 Segments with similar averages

[Ceballos *et al.*, 2018] describes a process on how to find windows of contiguous homozygosity,

Table 4.3: *Results of segmentation usign a non-penalized multivariate likelihood*

Segment no.	First index	Last index
1	1	15

i.e. segments in the genetic data in which the alleles are of the same type. This is of interest for a researching investigating diseases. Considering this problems scenario, Segmentr can be used to segment random variables $C_1, ..., C_n$ that represent genetic data, encoded as zero for homozygosity and one for heterozygosity, i.e. zero when the alleles are of the same type and one when the alleles are different.

So, in order to use Segmentr to solve this problem, it's necessary to find a likelihood function that favors segments the homogeneity of a given segment. That problem is approached by proposing a "mean" likelihood function, i.e. a function that maximizes the likelihood of segments whose elements approximate the segment average. One such function is defined in (4.2). Notice, however, the likelihood function proposed favors a single column segments, as it's logically the entity whose elements most approximate the average of the entire window. In order to counter this undesirable behavior, we penalized the function by subtracting from it a single constant, as described in (4.3). The constant factor has the effect of adding up when too many segments are considered in the estimation process, making it so wider segments end being picked up in the estimation process.

In order to observe how the proposed function behaves, consider the simple example defined in (4.4), in which X_i for $i \in \{1, \ldots, 20\}$ represent each a column indexed by i of a data set D, and Bern(p) representing the Bernoulli distribution with probability p. When segmenting the data set defined in (4.4) with the likelihood function defined in (4.3), we obtain the results displayed in Table 4.4.

$$L_\mu(X) = -\sum_i (x_i - E[X])^2 \;\Big|\; x_i \in X \tag{4.2}$$

$$P_{L_\mu}(X) = L_\mu(X) - 1 \tag{4.3}$$

Table 4.4: *Results of segmentation by aplying the likelihood function defined in* (4.3) *to a sample of size 10 of the model defined in* (4.4).

Segment no.	First index	Last index
1	1	5
2	6	15
3	16	20

$$
\begin{aligned}
X_1 &\sim \text{Bern}(0.9) \\
X_2 &\sim \text{Bern}(0.9) \\
X_3 &\sim \text{Bern}(0.9) \\
X_4 &\sim \text{Bern}(0.9) \\
X_5 &\sim \text{Bern}(0.9) \\
X_6 &\sim \text{Bern}(0.1) \\
X_7 &\sim \text{Bern}(0.1) \\
X_8 &\sim \text{Bern}(0.1) \\
X_9 &\sim \text{Bern}(0.1) \\
X_{10} &\sim \text{Bern}(0.1) \\
X_{11} &\sim \text{Bern}(0.1) \\
X_{12} &\sim \text{Bern}(0.1) \\
X_{13} &\sim \text{Bern}(0.1) \\
X_{14} &\sim \text{Bern}(0.1) \\
X_{15} &\sim \text{Bern}(0.1) \\
X_{16} &\sim \text{Bern}(0.9) \\
X_{17} &\sim \text{Bern}(0.9) \\
X_{18} &\sim \text{Bern}(0.9) \\
X_{19} &\sim \text{Bern}(0.9) \\
X_{20} &\sim \text{Bern}(0.9)
\end{aligned}
\tag{4.4}
$$

Chapter 5

Real Data Examples

In order to exemplify how Segmentr is used in real situations, an example on weather data is provided in this paper. The data was obtained using data found in [Wetterdienst, 2019].

Segmentr is a package that implements a handful of algorithms to segment a given data set, by finding the change points that maximize the collective likelihood of the segments according to an arbitrary likelihood function. So, the user of this package has to find an adequate likelihood for the segmentation problem to be solved, possibly having to penalize it in order to avoid either an overparametrized or underparametrized model, i.e. one with too many or too few change points, respectively. Also, it's important to consider the computation time of each algorithm and its trade-offs. This example walks rough the main concepts regarding its usage, using historical temperature data from Berlin as an example.

5.1 Understanding the data

The `berlin` data set, provided in this package, contains daily temperature measurements from 7 weather stations in Berlin for every day in the years 2010 and 2011, i.e., a total of 730 days. Therefore, every element in the data set has a temperature data point with units in Celsius, such that each of the 730 columns corresponds to a date, and each of the 7 rows corresponds to the weather station the data point was measured at. In Table 5.1, it's possible to see the first three columns, as well as the last three columns, together with the respective stations.

In order to grasp the behavior of the weather data, in Figure 5.1 the daily average temperature of all the weather stations, i.e., the mean value of each column in the data set as a time series graph in order to observe how the average temperature of Berlin behaves over time.

In the graph, the daily temperatures points alternate in upwards and downwards trends, which suggests it's possible to fit linear regressions for each of the upwards or downwards trend segments. So, a "linear regression likelihood" function is proposed, which should return a higher value the better a linear regression fits in a given segment. By intuition, we expect the likelihood function to segment the dataset approximately to the way the vertical lines are placed in Figure 5.2. The indices picked to represent each segment are available in Table 5.2.

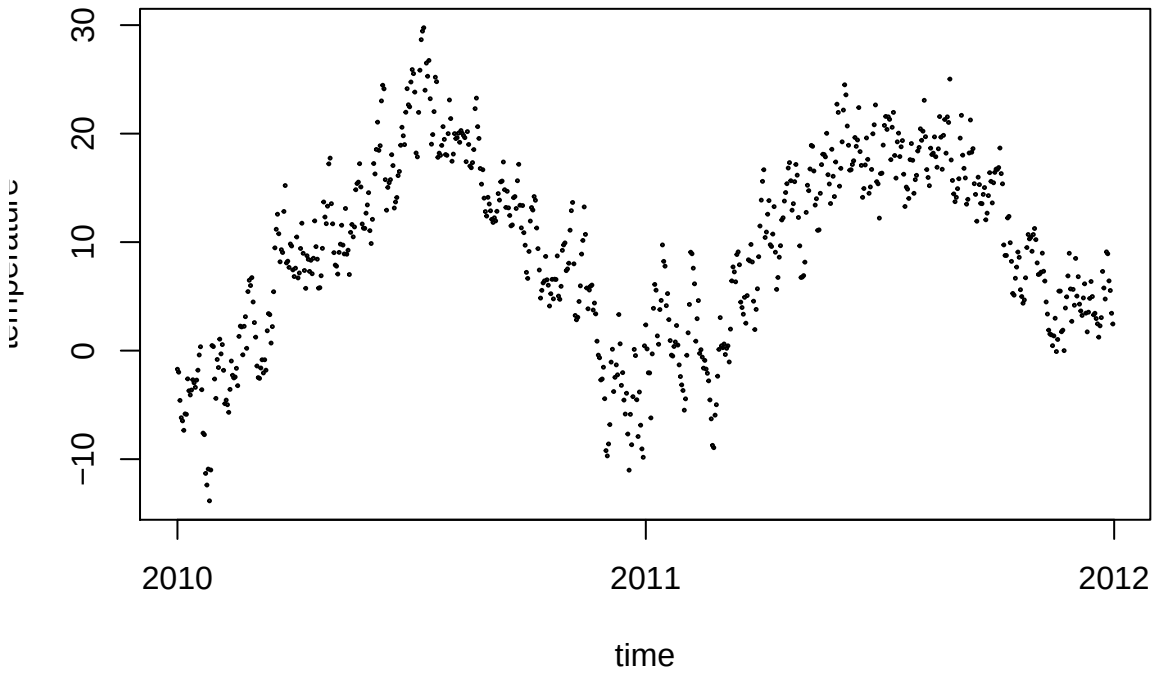

Figure 5.1: *Average daily temperature over time as measured by different stations in Berlin*

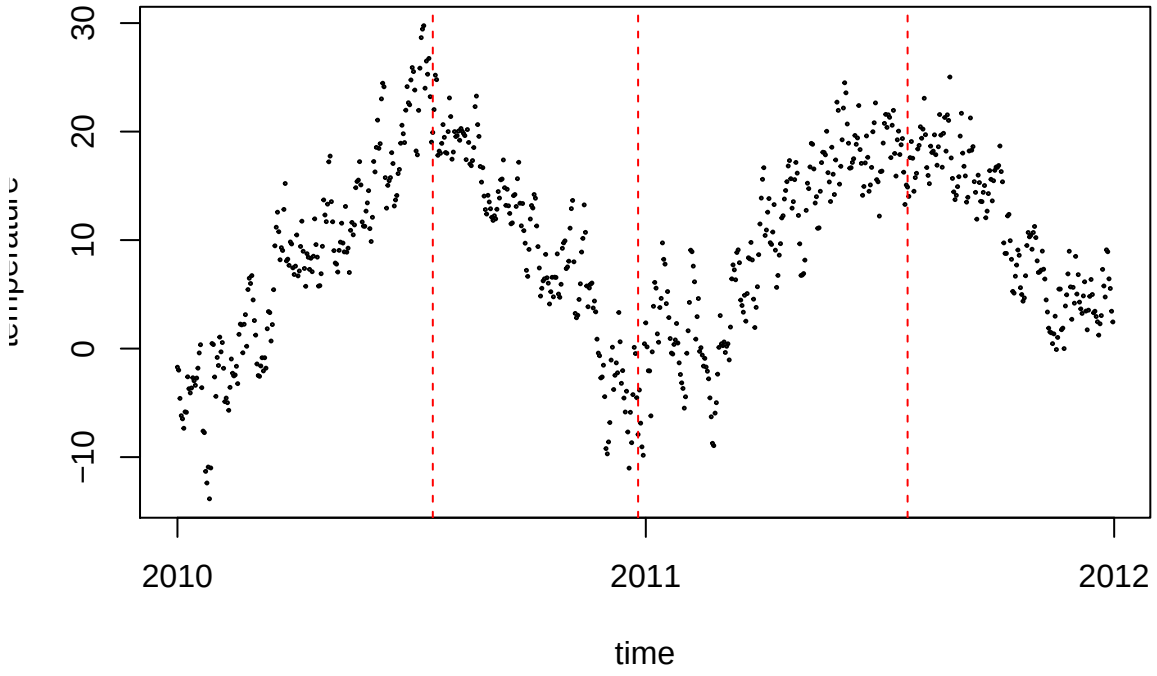

Figure 5.2: *Manual segmentation of the weather data picked according to intuition*

Table 5.1: *First and last columns of the 'berlin' dataset*

station	2010-01-01	2010-01-02	2010-01-03	..	2011-12-29	2011-12-30	2011-12-31
Berlin-Buch	-1.6	-1.9	-4.8	..	6.0	3.5	2.1
Berlin-Dahlem (FU)	-1.9	-2.3	-4.5	..	5.2	3.1	2.4
Berlin-Kaniswall	-1.9	-2.0	-4.6	..	5.2	3.3	1.9
Berlin-Marzahn	-1.8	-1.9	-4.8	..	5.7	3.7	2.6
Berlin-Schoenefeld	-1.9	-2.1	-4.6	..	5.0	3.3	2.4
Berlin-Tegel	-1.5	-2.0	-4.5	..	6.0	3.5	2.8
Berlin-Tempelhof	-1.4	-1.7	-4.3	..	5.7	3.7	3.0

Table 5.2: *Expected values to be found when segmenting the Berlin dataset*

Segment no.	First index	Last index
1	1	199
2	360	569
3	570	730

5.2 Building the likelihood

An adequate likelihood function should be able to rank a given set of possible segments and pick the best one given the evaluation criteria. Since the goal is to select segments with a good linear regression fit, the method's own standard log-likelihood function, i.e. the negative of the squared sum of residuals, is a good candidate for our segment likelihood function. However, the sum of residuals tend to increase with the amount of points in a segment. Instead, we pick the negative mean of the squared residuals, because it normalizes the likelihoods based on the length of the segment. So, in Equation (5.1), L_{lm} is the linear regression likelihood function, X is the set of points that belong to the segment, x_i and y_i are the points that belong to X for each index i, and f is the best linear regression that fitted the X segment.

$$L_{lm}(X) = -\sum_{i=1}^{n} \frac{1}{n}(y_i - f(x_i))^2 \tag{5.1}$$

A likelihood argument for a `segment()` call requires a function which accepts a candidate segment matrix, i.e. a subset of the columns in the data set, and returns the estimated likelihood for the candidate. Therefore, equation (5.1) is implemented as an R function called `lm_likelihood`, such that it obeys the argument contract by taking a matrix as argument and returning the negative mean of the squared residuals of a linear regression over the candidate segment. In order to get a sense of how the function behaves with the `berlin` data set, sample likelihoods a small, a medium, and a large segment are provided in 5.3.

With the likelihood function defined, it can now be applied to `segment()` in order to get

Table 5.3: *Sample values of the linear regression likelihood function for different sizes of segment candidates*

Small Size	Medium Size	Large Size
-0.0283673	-12.18277	-68.85185

Table 5.4: *Results of the segmentation algorithm using a non-penalized likelihood function*

Segment no.	First index	Last index
1	1	2
2	3	6
3	7	17
4	18	19
5	20	21
6	22	23
7	24	24
8	25	26
9	27	28
10	29	29
11	30	31
12	32	33
13	34	35
14	36	37
15	38	38
16	39	41
17	42	43
18	44	47
19	48	730

the segments for the data set. Since the time complexity of the exact algorithm is $O(n^2)$ and the number of points in the data set is high, the execution time required for the computation is quite prohibitive. So, for demonstration purposes, we use the hierarchical algorithm, due to its $O(n \log(n))$. We point the hierarchical algorithm, generally suitable for the multivariate likelihood, assumes the segments' likelihoods to be structured in a hierarchical manner, with a combination of two neighboring segments having being selected as an intermediate step before evaluating the ideal change points of the data set. The segmentation results can be seen in Table 5.4, and they are also plotted together with the weather data in Figure 5.3.

In Figure 5.3, it's possible see many very short segments, and a very large last segment. This is a result of the algorithm used, as well as the fact the `lm_likelihood` function tends to favor very short segments, as they usually have smaller residual error. So, to not get segments too short or too long, it's necessary to penalize the likelihood function for either extremely short or extremely long lengths.

5.3 Penalizing the likelihood function

To penalize a likelihood function in the Segmentr context is to decrease the return value of the likelihood function whenever unwanted segments are provided as an input. Typically, this involves penalizing the likelihood function whenever a very short or a very long segment is provided.

One such method is to subtract the output of the likelihood function with a penalty function

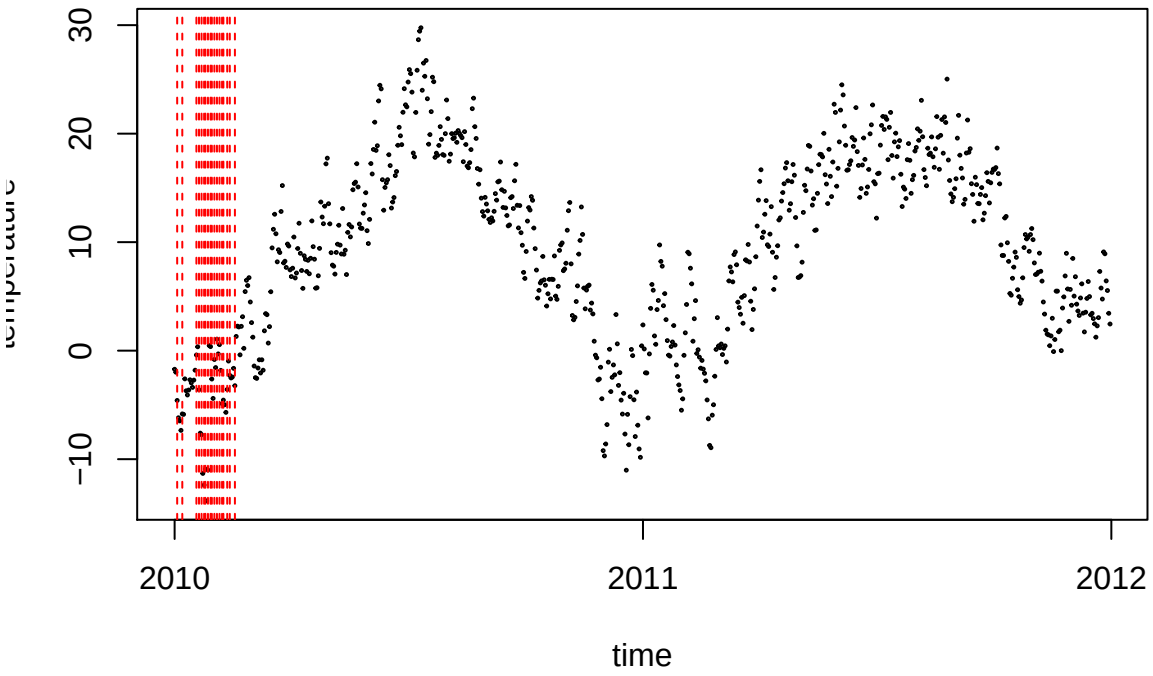

Figure 5.3: *Berlin weather data segmented using a non-penalized likelihood function*

which is a function of the length of the segment. We propose a penalty function in (5.2).

$$p(l) = C_1 e^{s_1(l-\frac{L}{2})} + C_2 e^{s_2(-l+\frac{L}{2})} \tag{5.2}$$

In equation (5.2), the penalty $p(l)$ is a function of the segment's length l, has the property that, for parametrization values $C_1 > 0$, $s_1 > 0$, $C_2 > 0$ and $s_2 > 0$, the penalty is high for values of l neighboring 0, as well as values of l the total length L of the data set. However, penalty is close to its minimum for values of l neighboring $\frac{L}{2}$. To visualize, consider a sample penalty function, with $C_1 = C_2 = 1$, $s_1 = s_2 = 0.3$ and $L = 100$, plotted in Figure 5.4.

Given the penalty function general formula, it's necessary to adjust the parameters such the penalty function has a scale compatible with the likelihood function. The `auto_penalize()` function, provided in the Segmentr package, builds a penalized version of the likelihood function, by estimating parametrization values based on sample likelihood values for big and small segments of the data set provided. The estimated parameters are tunable by adjusting the `small_segment_penalty` and the `big_segment_penalty` parameters, depending on how much small or big segments, respectively, should be penalized, i.e. the higher the parameter, the more penalized the related type of segment size is.

Let P_s be the `small_segment_penalty`, P_b be the `big_segment_penalty`, μ_s be the average likelihood for the sampled small segments and μ_b be the average likelihood for the samples big segments. The relationship between the parameters, as defined by the `auto_penalize()` function, is defined in (5.3).

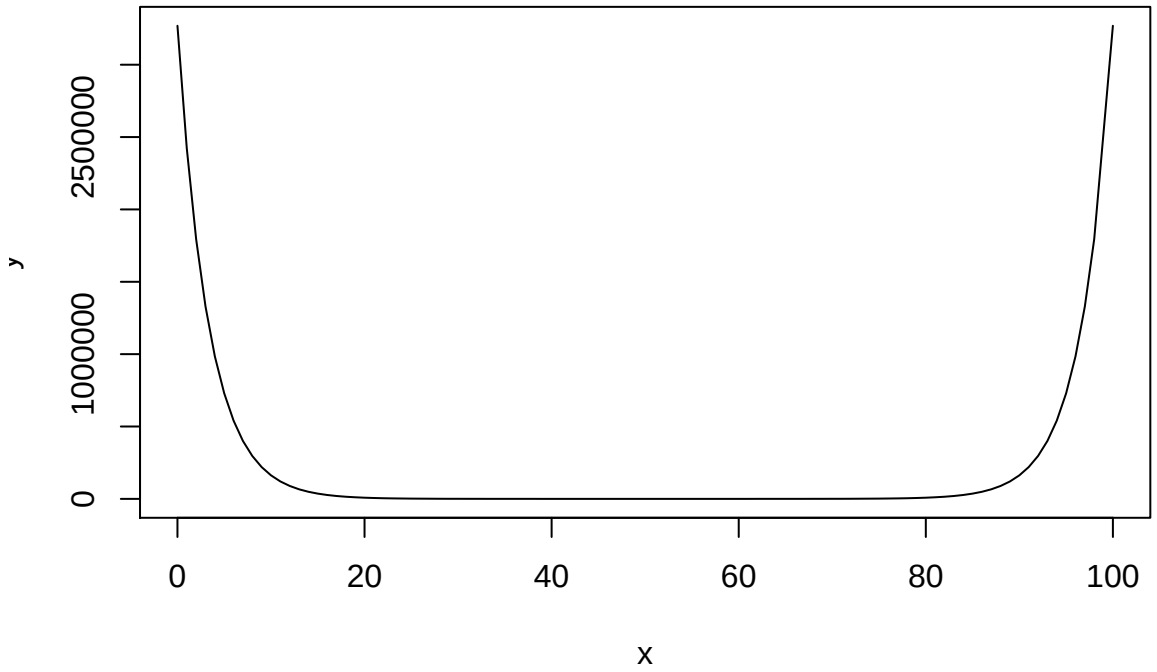

Figure 5.4: *Example curve of the penalty function for $C_1 = C_2 = 1$, $s_1 = s_2 = 0.3$ and $L = 100$*

Table 5.5: *Results of the segmentation algorithm using an auto-penalized likelihood function*

Segment no.	First index	Last index
1	1	203
2	204	268
3	269	327
4	328	559
5	560	644
6	645	730

$$\begin{aligned} C_1 &= \frac{\mu_b}{P_b} \\ s_1 &= \frac{4\log(P_b)}{L} \\ C_2 &= \frac{\mu_s}{P_s} \\ s_2 &= \frac{4\log(P_s)}{L} \end{aligned} \tag{5.3}$$

So, a penalized likelihood version of the function is created with `auto_penalize()` and then used with `segment()`. The results of the segmentation can be seen in Table 5.5, and it's possible to the segments with the weather data in Figure 5.5.

The function above found two segments, the last of which was still quite large, which suggests the `big_segment_penalty` argument needs to be increased to avoid that type of segment in the function. So, by increasing it's value, from the default of 10, all the way up to 100, we run the results again with the `segment()` function. The results can be seen in 5.6, and it's possible to the segments with the weather data in Figure 5.6.

Even with the new adjusted `penalized_likelihood`, the segments seen in Figure 5.6 aren't similar to what we are looking for as shown in Figure 5.2. The reason behind this, as discussed

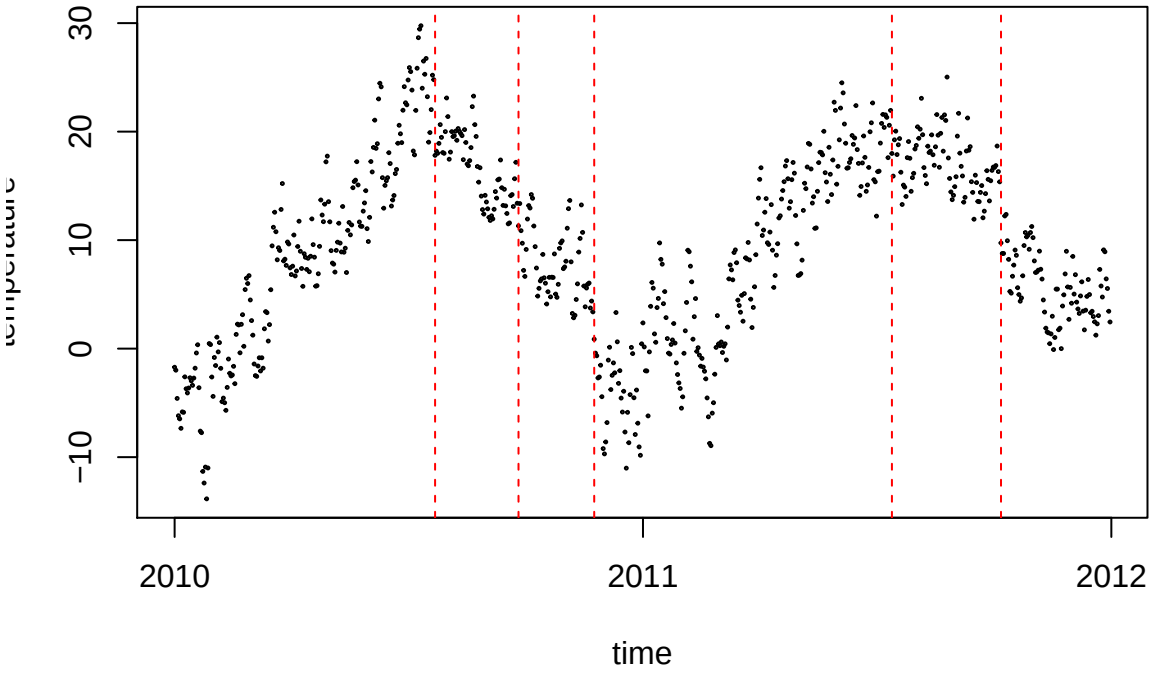

Figure 5.5: *Berlin weather data segmented using an auto-penalized likelihood function*

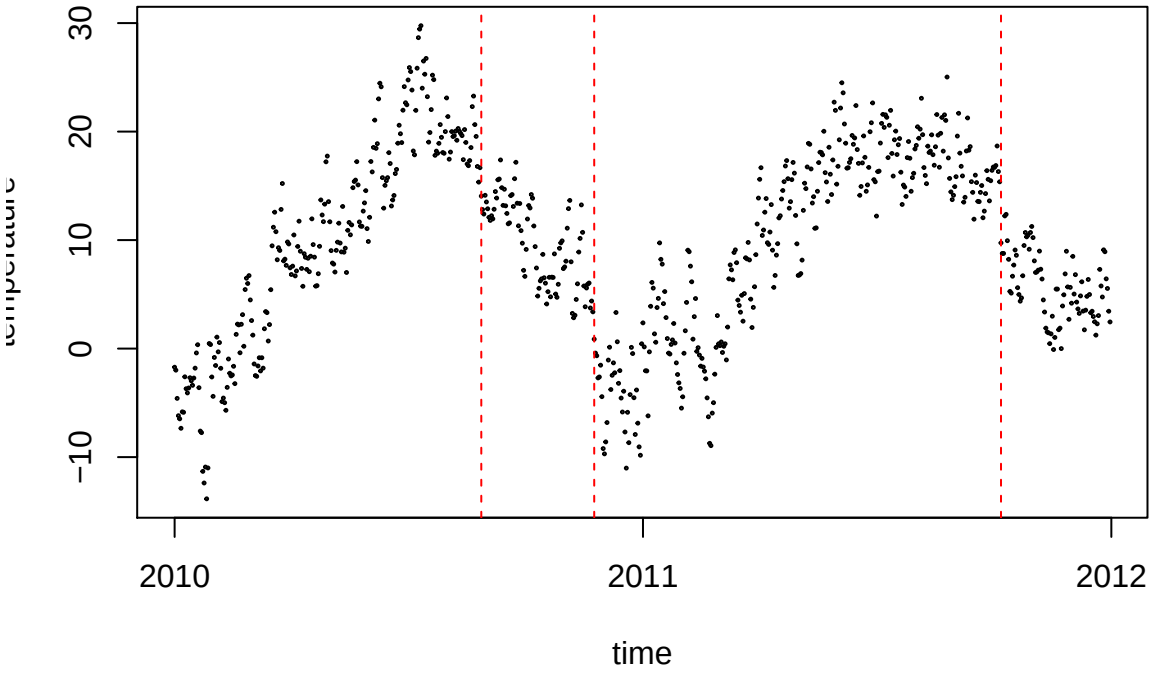

Figure 5.6: *Berlin weather data segmented using an adjusted penalized likelihood function*

Table 5.6: *Results of the segmentation algorithm using an adjusted penalized likelihood function*

Segment no.	First index	Last index
1	1	239
2	240	327
3	328	644
4	645	730

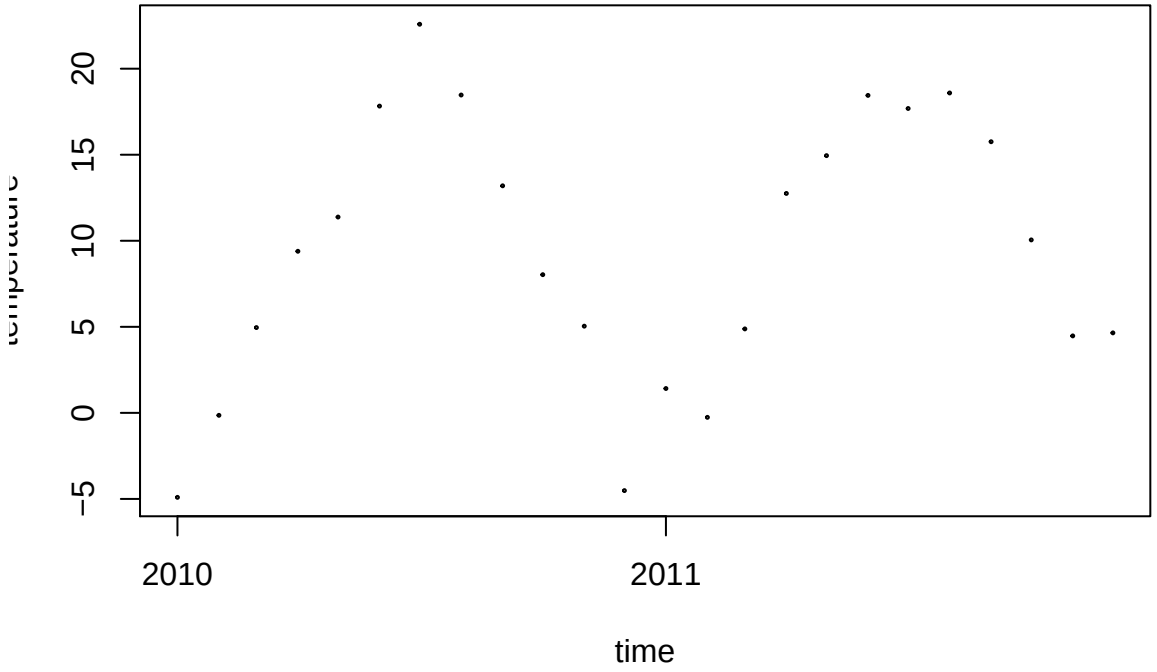

Figure 5.7: *Average monthly temperature over time as measured by different stations in Berlin*

earlier. So, in order to segment the data ideally, it's necessary to evaluate all of the possibilities.

The exact algorithm does precisely compute all of the possibilities, but its $O(n^2)$ time complexity is quite prohibitive to run the computation on the entire data set. So we can actually make the computation tolerable by reducing the granularity of the data, getting the monthly averages for each of the measurement stations.

5.4 Reducing granularity

The data set needs to be resampled in order to represent the monthly weather averages. It is done by computing the average temperature for each combination of month and weather station. With the granularity reduction, the data set will have 24 columns, one for each month in the two years period comprehended. The plot of the monthly average temperature of the temperatures of all the data points is plotted in figure 5.7

A new `penalized_function` is then built and applied to the monthly data set using `segment()`, with the results being shown in Table 5.7, and with we also plot it with weather data in Figure 5.8.

In order to make the solution comparison clearer, it's possible to rescale the `exact` solution to the monthly sampled dataset to the daily sampled dataset by applying a linear transformation of multiplying all the change points for the number of columns in the daily dataset and dividing it by the number of columns in the monthly dataset, i.e. each rescaled change point C_i relates to

Table 5.7: *Results of the segmentation algorithm using an auto-penalized likelihood function over downsampled berlin data*

Segment no.	First index	Last index
1	1	6
2	7	11
3	12	18
4	19	24

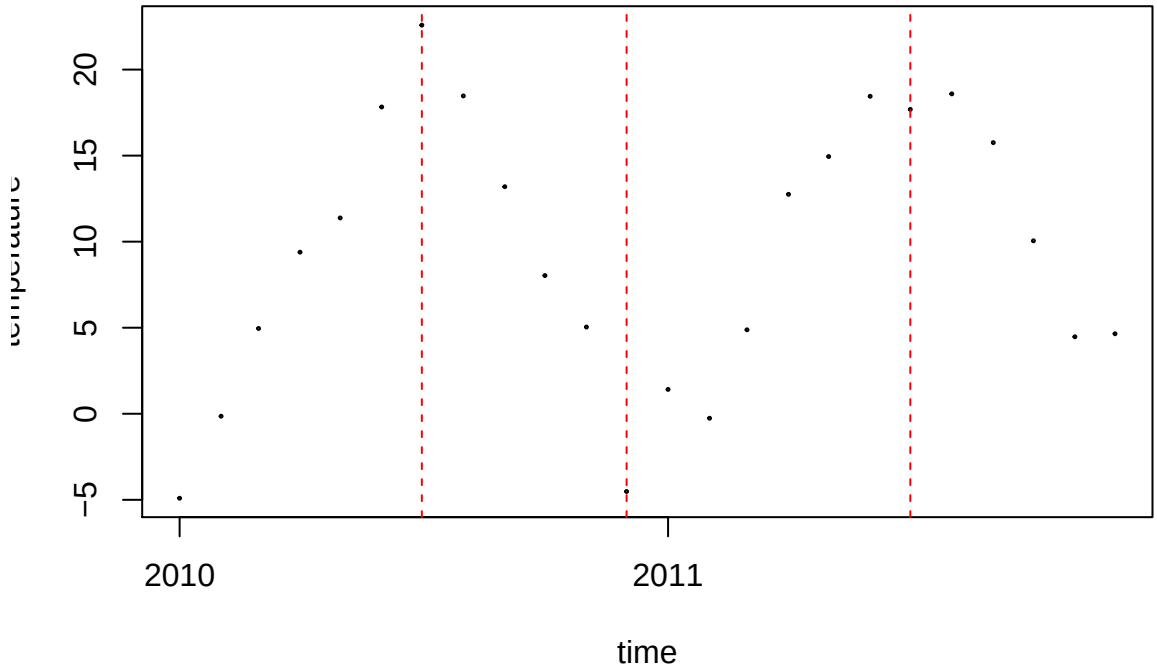

Figure 5.8: *Downsampled Berlin monthly weather data segmented using an auto penalized likelihood function*

the original change point c_i as described in equation (5.4), in which i is the index of each change point estimated with the monthly data set, N_M represents the number of columns in the monthly Berlin weather data set and N_B represents the number of columns in the daily Berlin data set, in the original form provided by the Segmentr package.

$$C_i = \left\lfloor c_i \frac{N_B}{N_M} \right\rfloor \tag{5.4}$$

After applying that transformation, the results can be seen in Table 5.8, as well as the plot along with the daily graph in Figure 5.9

With the exact solution, made possible with the granularity reduction, we noticed the data set is segmented closer to what the ideal solution would be, as the algorithm as able to evaluate all of the possibilities.

5.5 A non-usual likelihood function

Take notice the picking of the likelihood just have be able to rank segments in a desired manner. Therefore, there's freedom to pick a non-conventional likelihood function. For example, the R-squared statistic of the linear regression in each segment is conventionally used to infer how well the linear model fits the points in the data. It ranges from zero to one and the closer it is to

Table 5.8: *Results of the segmentation algorithm using an auto-penalized likelihood function over downsampled berlin data, but rescaled to fit the original data dimensions*

Segment no.	First index	Last index
1	1	212
2	213	364
3	365	577
4	578	731

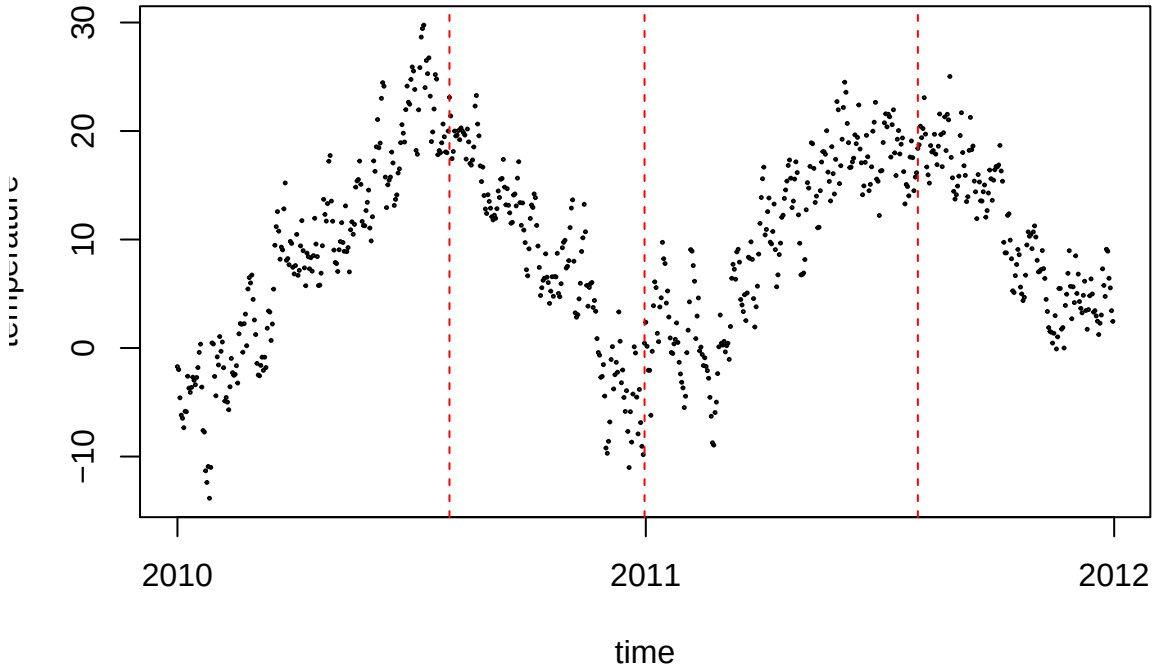

Figure 5.9: *Rescaled segments estimated for a downsampled monthly weather data of the Berlin data set using an auto penalized likelihood function*

one, the better it predicts the points in the data. Therefore, we propose the implementation of a `rsquared_likelihood`, which fits a linear regression against the data set received as input in the function, and then returns the R-squared statistic as the "likelihood" value of the segment. The defined function is applied against different segment sizes of the `berlin` data set, analogous to Table 5.3, and the results are shown in 5.9.

From what is observed in Table 5.9, and similar to the previous case, the new `rsquared_likelihood` has the highest values for small segments. Therefore, it needs to be penalized with the `auto_penalize` function. We then go on and apply the results of the penalized function and show them in Table 5.10. The plot with the weather data can be seen in Figure 5.10.

The default penalized `rsquared_likelihood` split the data set in three segments, roughly the same size each. Since the penalty model applied by `auto_penalize`, it applies the least penalty for values segments closer to about half the total length, so increasing the `big_penalty_segment` will have no effect. In fact, smaller segments are being over penalized. Because of this, it's necessary to reduce the `small_segment_penalty` segment, which we decrease to 1.5, from the default of 10. The results of the new segmentation can be seen in Table 5.11, and the plot in Figure 5.11.

With the adjusted parameters, we see the `rsquared_penalized` was able to segment the data in Figure 5.11 in a seemingly accurate manner, despite the nature of the hierarchical algorithm. As

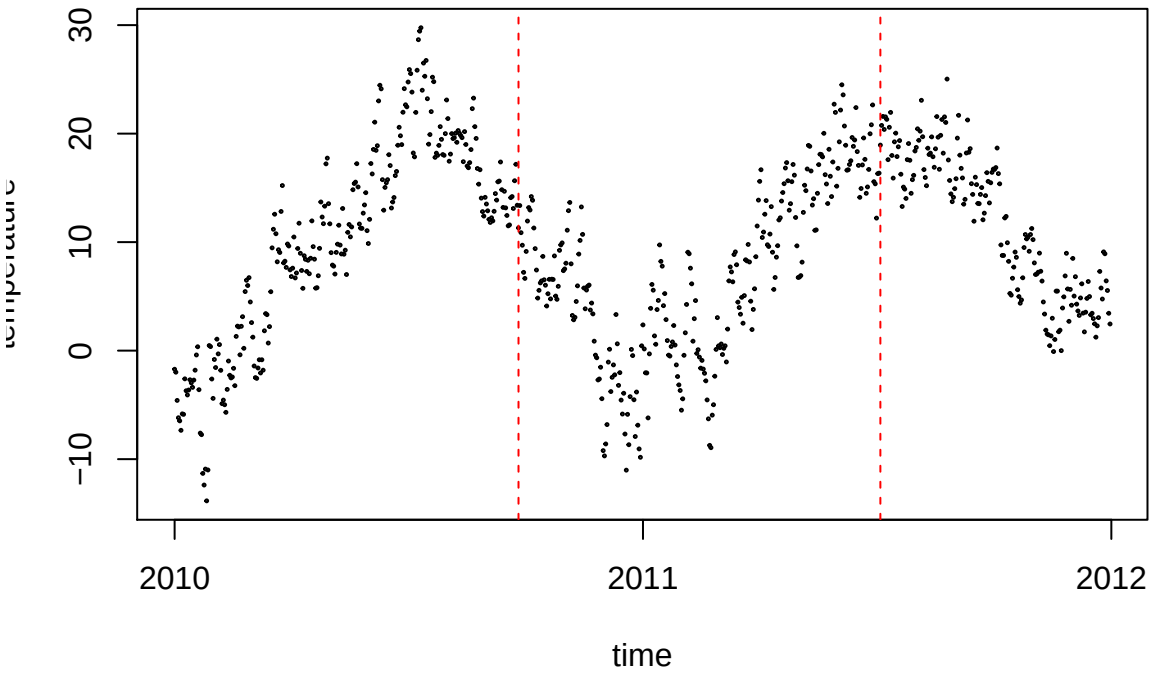

Figure 5.10: *Berlin daily weather data segmented using an auto-penalized R-squared likelihood function*

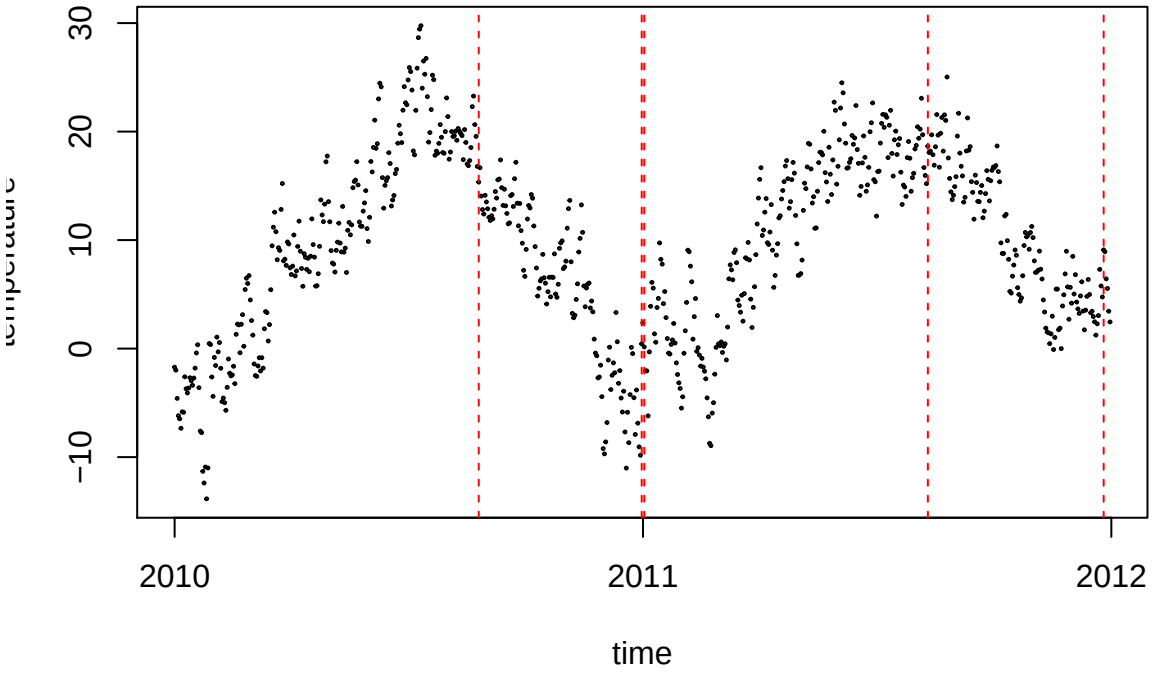

Figure 5.11: *Berlin daily weather data segmented using an adjusted penalized R-squared likelihood function*

Table 5.9: *Sample values of the R-squared likelihood function for different sizes of segment candidates*

Small Size	Medium Size	Large Size
0.982116	0.7601494	0.0328304

Table 5.10: *Results of the segmentation algorithm using an auto-penalized R-squared likelihood function over Berlin data*

Segment no.	First index	Last index
1	1	268
2	269	550
3	551	730

discussed previously, we point the hierarchical algorithm is not adequate for the linear likelihood, as the grouping of neighboring segments under a macro-segment, an intermediate segment state under the hierarchical algorithm, has a unattractive likelihood, due to the presence of alternating trends. Because of this, the macro-segment is never picked by the algorithm during the computation process, and so the ideal change points are never picked at their ideal positions. In order to see it more clearly, take the first 547 data points of the Berlin data set, roughly one year and a half. The results of the segmentation with that portion of the `berlin` data set can be seen in Table 5.12, and the plot can be seen in Figure 5.12.

In Figure 5.12, we can clearly see the segments do not match our expectations, as the algorithm is not able to bisect a macro-segment before finding the ideal segments in the next algorithm iteration.

Table 5.11: *Results of the segmentation algorithm using an adjusted penalized R-squared likelihood function over Berlin data*

Segment no.	First index	Last index
1	1	237
2	238	364
3	365	366
4	367	587
5	588	724
6	725	730

Table 5.12: *Results of the segmentation algorithm using an auto-penalized R-squared likelihood function over a subset of Berlin data*

Segment no.	First index	Last index
1	1	178
2	179	282
3	283	547

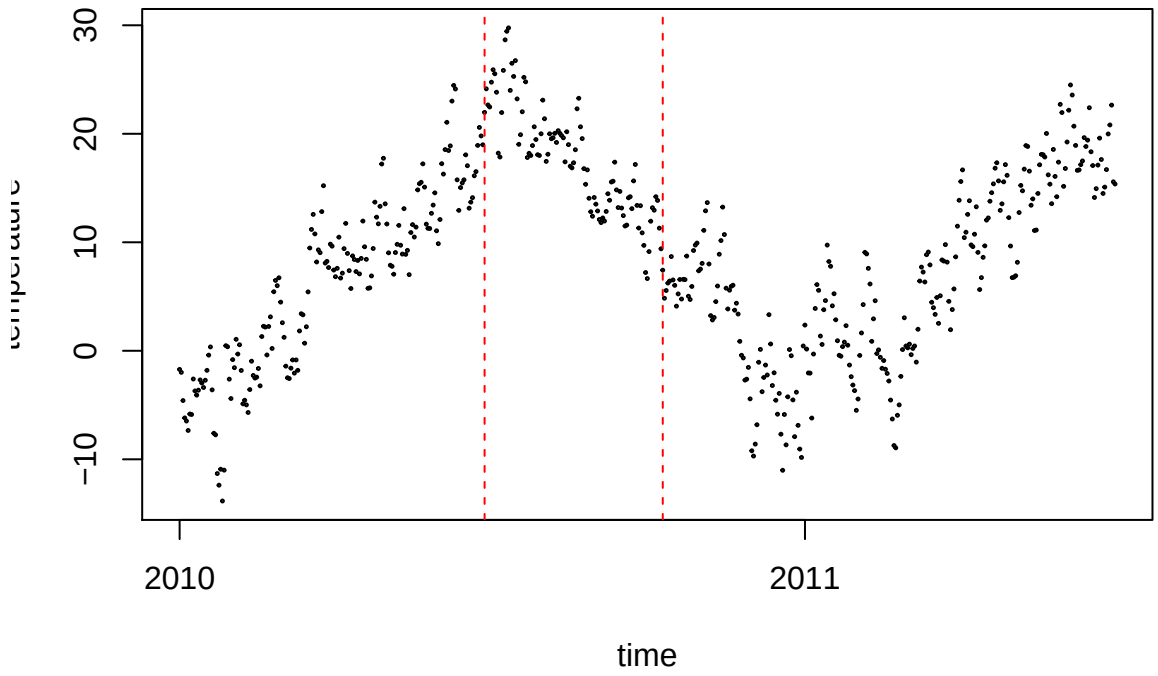

Figure 5.12: *Subset of Berlin daily weather data segmented using an auto-penalized R-squared likelihood function*

Chapter 6

Considerations on performance

This code shows a handful of attempts that were made in actually trying to execution time of the algorithms implemented by the package.

6.1 Benchmark the code

The package `microbenchmark` allows us to execute the code and compare different run-times. Therefore it's going to be used in this package to compare the performance of different code snippets.

6.2 Native Code

In the algorithms chapter, we discussed the time complexity of the different types of algorithms. In that scenario, the most relevant variable is the number of columns in the data set, which dictates whether the computation of the exact algorithm will be prohibitive or not. However, the number of samples does influence the computation time linearly. Given the most performed operation in any of the algorithms described in the chapter is the likelihood function, it's in essence the bottleneck of the whole computation. Therefore, real performance gains can be obtained depending on the performance of the application.

In R statistical programming, it's common place to use the functions provided by the programming environment, which are usually performant algorithms implemented in compiled programming languages such as C. However, there are times the programmer must implement a custom function in R, which in turn is a dynamically, and typically slower execution environment.

Therefore, in a first attempt to make the code execution run faster, and compare different results, Segmentr implements the [multivariate()] likelihood function in compiled C++ programming language, as well as it implements [r_multivariate()] which implements the same algorithm in the R programming environment.

In the scenario above, we notice performance improvements in the native [multivariate()] function, with the biggest performance improvement taking place in the `exact` algorithm results. Therefore, whenever the resources are available, it's recommended to implement the likelihood function directly in a native programming language such as C++. We recommend the RCpp package, which makes it easy to implement functions which interface easily with the R programming environment.

Table 6.1: *Execution time comparison between native C++ and interpreted R code*

expr	time (ms)
segment(data, likelihood = r_multivariate, algorithm = "hierarchical")	103.448
segment(data, likelihood = multivariate, algorithm = "hierarchical")	32.882
segment(data, likelihood = r_multivariate, algorithm = "exact")	50396.120
segment(data, likelihood = r_multivariate, algorithm = "hierarchical")	85.754
segment(data, likelihood = r_multivariate, algorithm = "exact")	48899.801
segment(data, likelihood = multivariate, algorithm = "hierarchical")	28.323
segment(data, likelihood = multivariate, algorithm = "exact")	7064.435
segment(data, likelihood = multivariate, algorithm = "exact")	7052.716

Table 6.2: *Execution time comparison between parralel and single-threaded computation with a data set of 100 samples and 10 columns*

expr	time (ms)
segment(data, likelihood = multivariate, algorithm = "hierarchical", allow_parallel = TRUE)	33.724
segment(data, likelihood = multivariate, algorithm = "exact", allow_parallel = TRUE)	244.178
segment(data, likelihood = multivariate, algorithm = "exact", allow_parallel = FALSE)	126.976
segment(data, likelihood = multivariate, algorithm = "hierarchical", allow_parallel = FALSE)	13.948

6.3 Parallelization

Given the increasing trend of parallel programming in the past few years, it's natural to try to take advantage of that by parallelizing the algorithm execution. In the algorithms implemented in this paper, parallelization is a bit complicated because a lot of the execution steps in the algorithm iterations depend on values computed in previous steps, and the golden rule of parallel computing is that faster results are generally obtained when computation steps are independent from one another, i.e. no dependency between the task, hence they can be performed in parallel. Yet, there are a few steps of the computation that can be performed in parallel, and its parallelization was implemented with the help of the `foreach` package.

Therefore, whether or not the computation will be performed depends on whether there is a parallel cluster registered in the R session with packages such as `doMC`, and whether the `allow_parallel` argument is true or false. With that in mind, a few comparisons are evaluated.

Analyzing the results above, we notice good improvements, especially for the "exact" algorithm. However, in a situation in which the number of rows isn't very large in comparison with the number of columns, the results of the parallel computation are a lot more modest. Consider the following case.

In the result above, parallelization achieved the opposite effect, actually making the computation slower, as there is now a lot of work done synchronizing necessary information between all the processes in the cluster. That takes more time than is saved by running the algorithm in parallel.

Taking all of that in consideration, the decision of whether or not to use parallelization relies

Table 6.3: *Execution time comparison between parralel and single-threaded computation with a data set of 5 samples and 100 columns*

expr	time (ms)
segment(data, likelihood = multivariate, algorithm = "exact", allow_parallel = FALSE)	5570.644
segment(data, likelihood = multivariate, algorithm = "hierarchical", allow_parallel = FALSE)	33.040
segment(data, likelihood = multivariate, algorithm = "exact", allow_parallel = TRUE)	6309.818
segment(data, likelihood = multivariate, algorithm = "hierarchical", allow_parallel = TRUE)	40.560

on the user after evaluating the nature of the data set. For data sets with few samples and many columns, parallelization is not recommended. In the case there are lots of samples, parallelization might give better computation times.

An alternative not developed in this project would be to use a genetic algorithm approach to the minimal set estimation, which is a probabilistic process that converges to the result after enough computation time have passed. The main advantage of such method is that computations can be more easily parallelized. Such development might be explored in future developments of this package.

Chapter 7

Accuracy of algorithms

Given different solutions obtained by the `segment` function, it's often necessary to compare them with each other. For that, the Hausdorff distance, defined in Chapter 2.3, can be used to measure a distance between two sets of estimated change points.

7.1 Accuracy of estimates for the Berlin dataset

In Chapter 5 the data set of weather temperatures in Berlin was presented, and there was a walk through on different ways to use Segmentr to find a good estimate set of segments for it. Table 5.2 lists the considered the ideal solution to the problem, at the same time as different solutions with different tables were found in Tables 5.4, 5.5 and 5.7. Considering this, the Hausdorff distance can be used to measure how far each estimate is from the ideal solution described in Table 5.2. In Table 7.1 we can see the results of the comparison, in which we can see the distance of each estimate calculated.

In Table 7.1 we can see the distance to the ideal is gradually decreased more adequate parameters are used to compute the segment estimates for the segmentation estimate. This improvement reflects the progress that can be observed in gradually in Figures 5.3, 5.5, 5.6 and 5.9.

7.2 Accuracy of algorithms using different algorithms

We want to measure how well the different algorithms provided in the Segmentr package find the segments in the example correlated columns simulated data set described in (4.1). Since the simulation was arbitrarily built, we know what the exact solution to the segmentation problem is.

Therefore, in Table 7.2 we can see the comparison of different algorithms with different parameters. We notice the "exact" algorithm manages to properly match the expected change points solution, whereas the "hierarchical" algorithm find an extra segment, which causes the Hausdorff distance to be bigger than zero. The two "hybrid" algorithm cases are interesting in the sense it that it shows how to algorithm works by either adopting the exact or the hierarchical algorithm depending on the threshold and the size of the segment being analyzed. When the threshold argument is large, it applies the exact algorithm to the data set, whereas when the threshold is small, it applies the hierarchical algorithm instead.

Consider now the simulated data set of segments with similar averages described in equation (4.4). We do the same algorithm comparison with this data set in Table 7.3. In contrast with the

Table 7.1: *Hausdorff distance for different segmentation attepmts over the Berlin weather data set*

Results referenced	Estimated Changepoints	Hausdorff Distance to Ideal
Table 5.2	200, 360, 570	0
Table 5.4	3, 7, 18, 20, 22, 24, 25, 27, 29, 30, 32, 34, 36, 38, 39, 42, 44, 48	522
Table 5.5	204, 269, 328, 560, 645	75
Table 5.6	240, 328, 645	75
Table 5.8	213, 365, 578	13

Table 7.2: *Comparison of solutions of different algorithms to segmenting the data set described in* (4.1)*, measuring how far each solution is from the ideal solution using the Hausdorff distance.*

Description	Changepoints	Hausdorff Distance
Expected solution	6, 11	0
Exact algorithm estimate	6, 11	0
Hierarchical algorithm estimate	6, 9, 11	2
Hybrid algorithm estimate with threshold 50	6, 11	0
Hybrid algorithm estimate with threshold 4	6, 9, 11	2

previous experiment, all the algorithms manage to find the expected solution to the problem.

Table 7.3: *Comparison of solutions of different algorithms to segmenting the data set described in* (4.4)*, measuring how far each solution is from the ideal solution using the Hausdorff distance.*

Description	Changepoints	Hausdorff Distance
Expected solution	6, 16	0
Exact algorithm estimate	6, 16	0
Hierarchical algorithm estimate	6, 16	0
Hybrid algorithm estimate with threshold 50	6, 16	0
Hybrid algorithm estimate with threshold 4	6, 16	0

Chapter 8

Conclusion

This work describes the segmentation problem, provides a theoretical model to work with the segmentation of data, and goes on to describe a few algorithms to try to estimate the set of points that optimally segment a given data set, explaining the differences between them.

Then we proceed and describe the actually implementation of such algorithms in an actual R package, in an attempt to make its use easier for researchers. We provide a few applications with simulated data, and also provide a real analysis with a real weather data set. Finally, performance considerations are taken into consideration when trying to have the code run faster.

In summary, there are a lot of decisions involved in the problem of segmentation, such as picking the right likelihood function for the evaluation, which algorithm to pick, and whether or not parallelization can help in running the computation faster. By providing a common set of tools, we hope to make the life of the researcher easier.

Appendix A

Package Usage

The purpose of this work is too have the `segmentr` to be as accessible as possible. Because of that, extra care was put into making it easy to install and use.

A.1 Installation

Given the algorithms and their applications discussed in [Castro *et al.*, 2018], the Segmentr package for R is proposed to help researchers segment their data sets. Installation can be done using the default `install.packages` command, like shown below.

```
install.packages("segmentr")
```

A.2 Usage

The package can be used with [segment()]. It takes a `data` argument containing a bi-dimensional matrix in which the rows represent different samples and the columns represent the comprehension of the data set we wish to segment. The function also accepts a `algorithm` argument, which can be `exact`, `hierarchical` or `hybrid`, that specifies the type of algorithm will be used when exploring the data set. Finally, it's also necessary to specify a `likelihood` function as argument, as it is used to compare and pick segments that are better fit according to the given likelihood's chosen criteria.

```
library("segmentr")

data <- rbind(
  c(1, 1, 0, 0, 0, 1023, 134521, 12324),
  c(1, 1, 0, 0, 0, -20941, 1423, 14334),
  c(1, 1, 0, 0, 0, 2398439, 1254, 146324),
  c(1, 1, 0, 0, 0, 24134, 1, 15323),
  c(1, 1, 0, 0, 0, -231, 1256, 13445),
  c(1, 1, 0, 0, 0, 10000, 1121, 331)
)
```

```
segment(
  data,
  algorithm = "exact",
  likelihood = function(X) multivariate(X) - 0.01*exp(ncol(X))
)
```

```
## Segments (total of 6):
##
## 1:1
## 2:2
## 3:3
## 4:4
## 5:5
## 6:8
```

Also, a vignette version of Chapter 5 is provided, and the user of the package can open it directly in R with embedded code examples and follow along the analysis of the data set.

Appendix B

Support Code

```
library(knitr)
library(kableExtra)
library(segmentr)
library(tibble)
library(magrittr)
knitr::opts_chunk$set(
  cache=TRUE,
  echo=FALSE
)
source("helper.R")
n <- 100
X1 <- sample(1:2, n, replace = TRUE)
X2 <- sample(1:2, n, replace = TRUE)
X3 <- sample(1:2, n, replace = TRUE)
X4 <- sample(1:2, n, replace = TRUE)
X5 <- sample(1:2, n, replace = TRUE)
X6 <- sample(1:2, n, replace = TRUE)

C1 <- X1
C2 <- X1 - X2
C3 <- X2
C4 <- X1 + X2
C5 <- X1
C6 <- X3
C7 <- X3 - X4
C8 <- X4
C9 <- X3 + X4
C10 <- X3
C11 <- X5
C12 <- X5 - X6
C13 <- X6
```

```
C14 <- X5 + X6
C15 <- X5
D_example <- cbind(C1, C2, C3, C4, C5, C6, C7, C8,
           C9, C10, C11, C12, C13, C14, C15)
head(D_example) %>%
  kable(caption="Sample values of the correlated random
        variables defined in (4.1).")
penalized_multivariate = function(X) multivariate(X) - 2 ^ ncol(X)

results_multivariate_penalized <- segment(
  D_example,
  likelihood = penalized_multivariate,
  algorithm = "exact"
)

print_results_table(
  results_multivariate_penalized,
  caption="Segment tables as defined in
  (4.1)."
)
segment(D_example, likelihood = multivariate, algorithm = "exact") %>%
  print_results_table(
    caption="Results of segmentation usign a non-penalized
      multivariate likelihood")
mean_likelihood <- function(X) {
  mean_value <- mean(X, na.rm = T)
  if (is.na(mean_value)) {
    0
  } else {
    -sum((X - mean_value)^2)
  }
}

penalized_mean_likelihood <- function(X) mean_likelihood(X) - 1

make_segment <- function(n, p) {
  matrix(rbinom(100 * n, 1, p), nrow = 100)
}

D_genetic <- cbind(
  make_segment(5, 0.9),
  make_segment(10, 0.1),
  make_segment(5, 0.9)
```

```
)

segment(
  D_genetic,
  likelihood = penalized_mean_likelihood,
  algorithm = "hieralg"
) %>%
  print_results_table(
    caption="Results of segmentation by aplying the
      likelihood function defined in
      (4.3) to a sample of size
      10 of the model defined in (4.4).
    ")
library(segmentr)
library(tidyr)
library(tibble)
library(dplyr)
library(lubridate)
library(magrittr)
library(purrr)
library(knitr)
library(kableExtra)
data(berlin)
as_tibble(berlin, rownames="station") %>%
  mutate(`..`="..") %>%
  select(station, `2010-01-01`:`2010-01-03`,
         `..`, `2011-12-29`:`2011-12-31`) %>%
  kable(caption="
    First and last columns of the `berlin` dataset
  ") %>%
  column_spec(1, width="5em")
berlin %>%
  colMeans() %>%
  enframe("time", "temperature") %>%
  mutate_at(vars(time), ymd) %>%
  with(plot(time, temperature, cex=0.2))
expected_berlin <- list(
  changepoints=c(200, 360, 570),
  segments=list(1:199, 360:569, 570:ncol(berlin))
)

plot_results(expected_berlin, berlin)
print_results_table(
```

```
  expected_berlin,
  caption="Expected values to be found when segmenting the Berlin dataset"
)
lm_likelihood <- function (data) {
  fit <- t(data) %>%
    as_tibble() %>%
    rowid_to_column() %>%
    gather(station, temperature, -rowid) %>%
    with(lm(temperature ~ rowid))

  -mean(fit$residuals ^ 2)
}

tibble(
  `Small Size`=lm_likelihood(berlin[, 2:3]),
  `Medium Size`=lm_likelihood(berlin[, 1:150]),
  `Large Size`=lm_likelihood(berlin)
) %>%
  kable(
    caption="Sample values of the linear regression
    likelihood function for different sizes of segment
    candidates"
  )
results_non_penalized <- segment(
  berlin,
  likelihood = lm_likelihood,
  algorithm = "hierarchical"
)

print_results_table(
  results_non_penalized,
  caption="Results of the segmentation algorithm
  using a non-penalized likelihood function"
)
plot_results(results_non_penalized, berlin)

plot_curve(~ exp(0.3*(. - 50)) + exp(0.3 * (-. + 50)),
           from = 0, to = 100, type="l")
penalized_likelihood <- auto_penalize(berlin,
                                      lm_likelihood)
results_auto_penalized <- segment(
  berlin,
  likelihood = penalized_likelihood,
```

```
  algorithm = "hierarchical"
)

print_results_table(
  results_auto_penalized,
  caption="Results of the segmentation algorithm
  using an auto-penalized likelihood function"
)
plot_results(results_auto_penalized, berlin)
penalized_likelihood <- auto_penalize(
  berlin,
  lm_likelihood,
  big_segment_penalty = 1000
)
results_adjusted_penalized <- segment(
  berlin,
  likelihood = penalized_likelihood,
  algorithm = "hierarchical"
)
print_results_table(
  results_adjusted_penalized,
  caption="Results of the segmentation algorithm
  using an adjusted penalized likelihood function"
)
plot_results(results_adjusted_penalized, berlin)
monthly_berlin <- berlin %>%
  as_tibble(rownames = "station") %>%
  gather(time, temperature, -station) %>%
  mutate(month = floor_date(ymd(time), "month")) %>%
  group_by(station, month) %>%
  summarize(temperature = mean(temperature)) %>%
  spread(month, temperature) %>% {
    stations <- .$station
    result <- as.matrix(.[, -1])
    rownames(result) <- stations
    result
  }

monthly_berlin %>%
  colMeans() %>%
  enframe("time", "temperature") %>%
  mutate_at(vars(time), ymd) %>%
  with(plot(time, temperature, cex=0.2))
```

```
penalized_likelihood <- auto_penalize(
  monthly_berlin,
  lm_likelihood,
  small_segment_penalty = 100
)

results_downscaled <- segment(
  monthly_berlin,
  likelihood = penalized_likelihood,
  algorithm = "exact"
)

print_results_table(
  results_downscaled,
  caption="Results of the segmentation algorithm
    using an auto-penalized likelihood function over
    downsampled berlin data"
)
plot_results(results_downscaled, monthly_berlin)
rescaled_changepoints <- round(
  results_downscaled$changepoints
  * ncol(berlin) / ncol(monthly_berlin)
)

results_rescaled <- with_segments(
  changepoints=rescaled_changepoints,
  len=ncol(berlin)
)

print_results_table(
  results_rescaled,
  caption="Results of the segmentation algorithm
    using an auto-penalized likelihood function over
    downsampled berlin data, but rescaled to fit the
    original data dimensions"
)
plot_results(results_rescaled, berlin)
rsquared_likelihood <- function (data) {
  as_tibble(t(data)) %>%
    rowid_to_column() %>%
    gather(station, temperature, -rowid) %>%
    with(lm(temperature ~ rowid)) %>%
    summary %>%
```

```
    .$adj.r.squared
}

tibble(
  `Small Size`=rsquared_likelihood(berlin[, 2:3]),
  `Medium Size`=rsquared_likelihood(berlin[, 1:150]),
  `Large Size`=rsquared_likelihood(berlin)
) %>%
  kable(
    caption="Sample values of the R-squared
    likelihood function for different sizes of segment
    candidates"
  )
penalized_likelihood <- auto_penalize(
  berlin,
  rsquared_likelihood
)
results <- segment(
  berlin,
  likelihood = penalized_likelihood,
  algorithm = "hierarchical"
)

print_results_table(
  results,
  caption="Results of the segmentation algorithm
    using an auto-penalized R-squared likelihood function over
    Berlin data"
)
plot_results(results, berlin)
penalized_likelihood <- auto_penalize(
  berlin,
  rsquared_likelihood,
  small_segment_penalty = 1.5
)
results <- segment(
  berlin,
  likelihood = penalized_likelihood,
  algorithm = "hierarchical"
)
print_results_table(
  results,
  caption="Results of the segmentation algorithm
```

```
    using an adjusted penalized R-squared likelihood
    function over Berlin data"
)
plot_results(results, berlin)
sub_berlin <- berlin[, 1:547]
penalized_likelihood <- auto_penalize(
  sub_berlin,
  rsquared_likelihood
)
results <- segment(
  sub_berlin,
  likelihood = penalized_likelihood,
  algorithm = "hierarchical"
)
print_results_table(
  results,
  caption="Results of the segmentation algorithm
    using an auto-penalized R-squared likelihood
    function over a subset of Berlin data"
)
plot_results(results, sub_berlin)
source("helper.R")
library(microbenchmark)
data <- makeRandom(20, 100)

bench <- microbenchmark(
  segment(data,
          likelihood = multivariate,
          algorithm = "exact"),
  segment(data,
          likelihood = r_multivariate,
          algorithm = "exact"),
  segment(data,
          likelihood = multivariate,
          algorithm = "hierarchical"),
  segment(data,
          likelihood = r_multivariate,
          algorithm = "hierarchical"),
  times = 2
)

print_benchmark(
  bench,
```

```
  caption="Execution time comparison
    between native C++ and interpreted R code"
)
data <- makeRandom(100, 10)
doMC::registerDoMC(4)
bench <- microbenchmark(
  segment(data,
          likelihood = multivariate,
          algorithm = "exact",
          allow_parallel = FALSE),
  segment(data,
          likelihood = multivariate,
          algorithm = "exact",
          allow_parallel = TRUE),
  segment(data,
          likelihood = multivariate,
          algorithm = "hierarchical",
          allow_parallel = FALSE),
  segment(data,
          likelihood = multivariate,
          algorithm = "hierarchical",
          allow_parallel = TRUE),
  times = 1
)
print_benchmark(
  bench,
  caption="Execution time comparison
    between parralel and single-threaded computation
    with a data set of 100 samples and 10 columns"
)
data <- makeRandom(5, 100)
doMC::registerDoMC(4)
bench <- microbenchmark(
  segment(data,
          likelihood = multivariate,
          algorithm = "exact",
          allow_parallel = FALSE),
  segment(data,
          likelihood = multivariate,
          algorithm = "exact",
          allow_parallel = TRUE),
  segment(data,
          likelihood = multivariate,
```

```
          algorithm = "hierarchical",
          allow_parallel = FALSE),
  segment(data,
          likelihood = multivariate,
          algorithm = "hierarchical",
          allow_parallel = TRUE),
  times = 1
)

print_benchmark(
  bench,
  caption="Execution time comparison
    between parralel and single-threaded computation
    with a data set of 5 samples and 100 columns"
)
source("helper.R")
hausdorff <- function(set_a, set_b) {
  distance <- Vectorize(function(p1, p2) abs(p1 - p2))

  distance_1_to_2 <- function(set_1, set_2) {
    max(sapply(set_1, function(p1) min(distance(p1, set_2))))
  }

  max(distance_1_to_2(set_a, set_b), distance_1_to_2(set_b, set_a))
}

segment_distance <- function(changepoints1, changepoints2) {
  hausdorff(c(1, changepoints1), c(1, changepoints2))
}
deviation <- partial(
  segment_distance,
  changepoints2=expected_berlin$changepoints
)

tribble(
  ~`Results referenced`,
    ~`Estimated Changepoints`,
      ~`Hausdorff Distance to Ideal`,
  "Table 5.2",
    comma_format(expected_berlin$changepoints),
      deviation(expected_berlin$changepoints),
  "Table 5.4",
    comma_format(results_non_penalized$changepoints),
```

```
      deviation(results_non_penalized$changepoints),
  "Table 5.5",
    comma_format(results_auto_penalized$changepoints),
      deviation(results_auto_penalized$changepoints),
  "Table 5.6",
    comma_format(results_adjusted_penalized$changepoints),
      deviation(results_adjusted_penalized$changepoints),
  "Table 5.8",
    comma_format(results_rescaled$changepoints),
      deviation(results_rescaled$changepoints)
) %>% kable(
  caption="Hausdorff distance for different
  segmentation attepmts over the Berlin weather data set"
) %>% column_spec(2, width="15em")
expected_multivariate_example <- c(6, 11)

deviation <- partial(
  segment_distance,
  changepoints2=expected_multivariate_example
)

exact_multivariate_changepoints <- segment(
  D_example,
  likelihood = penalized_multivariate,
  algorithm = "exact"
)$changepoints

hieralg_multivariate_changepoints <- segment(
  D_example,
  likelihood = penalized_multivariate,
  algorithm = "hierarchical"
)$changepoints

hybrid_multivariate_changepoints <- segment(
  D_example,
  likelihood = penalized_multivariate,
  algorithm = "hybrid"
)$changepoints

hybrid_multivariate_changepoints_threshold <- segment(
  D_example,
  likelihood = penalized_multivariate,
  algorithm = "hybrid",
```

```
  threshold = 4
)$changepoints

tribble(
  ~`Description`,
    ~`Changepoints`,
      ~`Hausdorff Distance`,
  "Expected solution",
    comma_format(expected_multivariate_example),
      deviation(expected_multivariate_example),
  "Exact algorithm estimate",
    comma_format(exact_multivariate_changepoints),
      deviation(exact_multivariate_changepoints),
  "Hierarchical algorithm estimate",
    comma_format(hieralg_multivariate_changepoints),
      deviation(hieralg_multivariate_changepoints),
  "Hybrid algorithm estimate with threshold 50",
    comma_format(hybrid_multivariate_changepoints),
      deviation(hybrid_multivariate_changepoints),
  "Hybrid algorithm estimate with threshold 4",
    comma_format(hybrid_multivariate_changepoints_threshold),
      deviation(hybrid_multivariate_changepoints_threshold)
) %>%
  kable(caption="Comparison of solutions of different algorithms
to segmenting the data set described in (4.1),
measuring how far each solution is from the ideal
solution using the Hausdorff distance.")
expected_mean_example <- c(6, 16)

deviation <- partial(
  segment_distance,
  changepoints2=expected_mean_example
)

exact_mean_changepoints <- segment(
  D_genetic,
  likelihood = penalized_mean_likelihood,
  algorithm = "exact"
)$changepoints

hieralg_mean_changepoints <- segment(
  D_genetic,
  likelihood = penalized_mean_likelihood,
```

```
  algorithm = "hierarchical"
)$changepoints

hybrid_mean_changepoints <- segment(
  D_genetic,
  likelihood = penalized_mean_likelihood,
  algorithm = "hybrid"
)$changepoints

hybrid_mean_changepoints_threshold <- segment(
  D_genetic,
  likelihood = penalized_mean_likelihood,
  algorithm = "hybrid",
  threshold = 4
)$changepoints

tribble(
  ~`Description`,
    ~`Changepoints`,
      ~`Hausdorff Distance`,
  "Expected solution",
    comma_format(expected_mean_example),
      deviation(expected_mean_example),
  "Exact algorithm estimate",
    comma_format(exact_mean_changepoints),
      deviation(exact_mean_changepoints),
  "Hierarchical algorithm estimate",
    comma_format(hieralg_mean_changepoints),
      deviation(hieralg_mean_changepoints),
  "Hybrid algorithm estimate with threshold 50",
    comma_format(hybrid_mean_changepoints),
      deviation(hybrid_mean_changepoints),
  "Hybrid algorithm estimate with threshold 4",
    comma_format(hybrid_mean_changepoints_threshold),
      deviation(hybrid_mean_changepoints_threshold)
) %>%
  kable(caption="Comparison of solutions of different algorithms
to segmenting the data set described in (4.4),
measuring how far each solution is from the ideal
solution using the Hausdorff distance.")
install.packages("segmentr")
library("segmentr")
```

```
data <- rbind(
  c(1, 1, 0, 0, 0, 1023, 134521, 12324),
  c(1, 1, 0, 0, 0, -20941, 1423, 14334),
  c(1, 1, 0, 0, 0, 2398439, 1254, 146324),
  c(1, 1, 0, 0, 0, 24134, 1, 15323),
  c(1, 1, 0, 0, 0, -231, 1256, 13445),
  c(1, 1, 0, 0, 0, 10000, 1121, 331)
)

segment(
  data,
  algorithm = "exact",
  likelihood = function(X) multivariate(X) - 0.01*exp(ncol(X))
)
vignette("segmentr")
```

Bibliography

Castro *et al.***(2018)** Bruno M. Castro, Renan B. Lemes, Jonatas Cesar, Tábita Hünemeier e Florencia Leonardi. A model selection approach for multiple sequence segmentation and dimensionality reduction. *Journal of Multivariate Analysis*, 167:319 – 330. ISSN 0047-259X. doi: https://doi.org/10.1016/j.jmva.2018.05.006. URL http://www.sciencedirect.com/science/article/pii/S0047259X18302331. Citado na pág.

Ceballos *et al.***(2018)** Francisco Ceballos, Peter Joshi, David W. Clark, Michèle Ramsay e James F. Wilson. Runs of homozygosity: Windows into population history and trait architecture. *Nature Reviews Genetics*, 19. doi: 10.1038/nrg.2017.109. Citado na pág.

Eddelbuettel e François(2011) Dirk Eddelbuettel e Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18. doi: 10.18637/jss.v040.i08. URL http://www.jstatsoft.org/v40/i08/. Citado na pág.

Maidstone *et al.***(2017)** Robert Maidstone, Toby Hocking, Guillem Rigaill e Paul Fearnhead. On optimal multiple changepoint algorithms for large data. *Statistics and Computing*, 27(2): 519–533. ISSN 1573-1375. doi: 10.1007/s11222-016-9636-3. URL https://doi.org/10.1007/s11222-016-9636-3. Citado na pág.

Mello(2019) Thales Mello. Segmentr: An r package to search for independent segments in a sequence (aka change points). https://github.com/thalesmello/segmentr, 2019. Citado na pág.

Mello e Leonardi(2019) Thales Mello e Florencia Leonardi. *segmentr: Segment Data With Maximum Likelihood*, 2019. URL https://CRAN.R-project.org/package=segmentr. R package version 0.1.1. Citado na pág.

Munkres(2000) J.R. Munkres. *Topology*. Featured Titles for Topology Series. Prentice Hall, Incorporated. ISBN 9780131816299. URL https://books.google.com.br/books?id=XjoZAQAAIAAJ. Citado na pág.

Park(2017) K.I. Park. *Fundamentals of Probability and Stochastic Processes with Applications to Communications*. Springer International Publishing. ISBN 9783319680743. URL https://books.google.com.br/books?id=cd2SswEACAAJ. Citado na pág.

R Core Team(2018) R Core Team. R: A language and environment for statistical computing, 2018. URL https://www.R-project.org/. Citado na pág.

Wetterdienst(2019) Deutscher Wetterdienst. Climate data center - ftp server. https://www.dwd.de/EN/climate_environment/cdc/cdc.html, 2019. Citado na pág.

Wickham(2015) Hadley Wickham. *R Packages*. O'Reilly Media, Inc., 1st edição. ISBN 1491910593, 9781491910597. URL http://r-pkgs.had.co.nz/. Citado na pág.