Learning algorithms and statistical software, with applications to bioinformatics

Toby Dylan Hocking

THÈSE DE DOCTORAT
DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

présentée par **Toby Dylan Hocking**

pour obtenir le grade de
Docteur de l'École Normale Supérieure de Cachan

Domaine: **Mathématiques appliquées**

Sujet de la thèse:

Algorithmes d'apprentissage et logiciels pour la statistique, avec applications à la bioinformatique

—

Learning algorithms and statistical software, with applications to bioinformatics

---

Thèse présentée et soutenue à Paris le 20 novembre 2012
devant le jury composé de:

Francis BACH	ENS/INRIA Paris	Directeur de thèse
Jean-Philippe VERT	Mines ParisTech/Institut Curie	Directeur de thèse
Stéphane ROBIN	AgroParisTech	Rapporteur
Yves GRANDVALET	Université de Compiègne	Rapporteur
Idris ECKLEY	Lancaster University	Examinateur
Isabelle JANOUEIX-LEROSEY	Institut Curie	Examinateur

Thèse préparée au sein de l'équipe SIERRA
au département d'informatique de l'ENS Ulm
(INRIA/ENS/CNRS UMR 8548)
et l'équipe CBIO à Mines ParisTech, Institut Curie, INSERM U900.

Contents

Acknowledgements

For providing such a wonderful scientific environment to work in, I would like to thank all my colleagues at the Mines ParisTech Centre for Computational Biology, INRIA Sierra, and the Institute Curie bioinformatics group (INSERM u900). In particular, I would like to thank my advisors Jean-Philippe Vert and Francis Bach, and my coauthors Armand Joulin, Gudrun Schleiermacher, Isabelle Janoueix-Lerosey, Olivier Delattre, and Guillem Rigaill.

For supporting me while I was growing up and for coming to Paris to set up my PhD reception, I would like to thank my mom Lori Ann Hocking and my aunt Diana Stevens.

For supporting me financially during the three years of my doctoral studies, I would like to thank Digiteo.

Chapter 1

Introduction

In this introductory chapter, I first present a summary of the contributions of this thesis in French and English. Then, for the rest of this thesis I will use English. In the rest of the introduction, I present a brief review of the relevant biology, machine learning, and statistical software necessary to understand the contributions of this thesis.

This thesis focuses on the development of new mathematical models for data analysis. To show results, and to explain the geometric interpretation of the models, I will make extensive use of color figures. To integrate the figures with the discussion, I have tried to follow the guidelines of the Yale statistician Edward Tufte, who proposed to "completely integrate words, numbers, images, diagrams" as a sound principle for the analysis and presentation of data [Tufte, 2006, page 131]. So to assist understanding, I have tried to place the textual discussion of each figure on the same page as the figure itself. If that was not practical, I have placed the figure on the facing page. Also, this thesis contains some blank space and page breaks. Again, I added these to ensure that figures and tables appear near the text that discusses them.

The best way to read this thesis is in printed book form. When reading on a computer, I suggest a reader that supports "Dual" display where two pages are displayed at the same time. In particular, even numbered pages should appear on the left and odd numbered pages should appear on the right.

1.1 Résumé du travail

L'apprentissage statistique est le domaine des mathématiques qui aborde le développement des algorithmes d'analyse de données. Cette thèse est divisée en deux parties : la présentation de modèles mathématiques et l'implémentation d'outils logiciels.

Dans la première partie, je présente de nouveaux algorithmes pour la segmentation et pour le partitionnement de données (clustering). Le partitionnement de données et la segmentation sont des méthodes d'analyse qui cherche des structures dans les données. Je présente les contributions suivantes, en soulignant les applications à la bioinformatique.

- Dans le chapitre 2, je présente "clusterpath," un algorithme pour le partitionnement de données utilisant l'optimisation convexe.
- Dans le chapitre 3, je développe une méthode pour la séléction de modèle de segmentation qui exploite les annotations visuelles.
- Dans le chapitre 4, je détermine des penalités optimales pour la détection de changements dans les signaux homoscédastique et constant par morceaux.
- Dans le chapitre 5, j'explique comment apprendre une fonction de penalité à partir d'annotations visuelles.

Dans la deuxième partie, je présente mes contributions au logiciel libre pour la statistique, qui est utilisé pour l'analyse quotidienne du statisticien.

- Dans le chapitre 7, je présente le paquet R **directlabels**, qui remplace une légende par l'annotation directe dans les graphiques statistiques.
- Dans le chapitre 8, je présente le paquet R **inlinedocs**, qui permet d'écrire la documentation du code R dans les commentaires.
- Dans le chapitre 9, je présente une fonction qui permet l'utilisation de parenthèses de groupement pour les expressions rationelles, ce qui peut être utilisé pour le pré-traitement de données dans R-2.14.

1.2 Summary of contributions

Statistical machine learning is a branch of mathematics concerned with developing algorithms for data analysis. This thesis presents new mathematical models and statistical software for data analysis, and is organized into two parts.

In the first part, we present several new algorithms for clustering and segmentation. Clustering and segmentation are a class of techniques that attempt to find structures in data. We discuss the following contributions, with a focus on applications to cancer data from bioinformatics.

- In Chapter 2, we present "clusterpath," an algorithm for clustering that uses fusion penalties in a convex optimization problem. Clustering algorithms are useful for finding latent classes and tree structures in data, but classical algorithms like hierarchical clustering and k-means have some limitations. We propose a convex relaxation of hierarchical clustering, yielding the "clusterpath" which has a natural geometric interpretation. We give efficient algorithms for calculating the continuous regularization path of solutions, and discuss relative advantages of the model parameters. Our method experimentally gives state-of-the-art results similar to spectral clustering for non-convex clusters, and has the added benefit of learning a tree structure from the data. The free/open-source code is available in the **clusterpath** package on R-Forge.

- In Chapter 3, we explain a method for segmentation model selection that uses visual annotations to train and compare models in real data.

  Many models have been proposed to detect copy number alterations in chromosomal copy number profiles, but it is usually not obvious to decide which is most effective for a given data set. Furthermore, most methods have a smoothing parameter that determines the number of breakpoints and must be chosen using various heuristics.

  We present three contributions for copy number profile smoothing model selection. First, we propose to select the model and degree of smoothness that maximizes agreement with visual breakpoint region annotations. Second, we develop cross-validation procedures to estimate the error of the trained models. Third, we apply these methods to compare many existing models on a new database of annotated neuroblastoma copy number profiles, which we make available as a public benchmark for testing new algorithms. Whereas previous studies have

been qualitative or limited to simulated data, our approach is quantitative and suggests which algorithms are most accurate in practice on real data.

Several free/open-source software packages were developed in support of this chapter. The code that implements the model comparisons is available in the **bams** package on CRAN. Two GUIs for creating annotation databases are also available.

- In Chapter 4, we calculate optimal penalties for change-point detection in simulated piecewise constant signals. Given a latent piecewise constant signal, we define a precise breakpoint detection error function, and discuss its relationship to the annotation error defined in Chapter 3. We then use the error function to determine optimal penalties for breakpoint detection in databases of simulated signals of varying sampling density, noise, and length.

- In Chapter 5, we explain how to use visual annotations to learn an optimal penalty function for change-point detection in real data. In the previous chapters, we saw that the segmentation models selected using standard criteria do not accurately recover the change-points defined in databases of visual annotations. So to find a model that agrees with the annotations, we propose to learn a penalty function that minimizes the non-convex annotation error. We propose a convex relaxation that yields an interval regression problem, and solve it using accelated proximal gradient methods. Finally, we show that this method achieves state-of-the-art performance on several annotation data sets based on the neuroblastoma data.

In the second part, we focus on statistical software contributions which we implemented in the R progamming language, and are practical for use in everyday data analysis.

- In Chapter 7, we present the R package **directlabels**, which replaces confusing legends with direct labels in statistical graphics. Direct labels are often much easier to interpret than a legend, but are not often used in practice since the label positions must be determined based on the data. We present several algorithms that can be used to find readable direct labels for several common plot types, and discuss the design and implementation of the **directlabels** package.

- In Chapter 8, we present the R package **inlinedocs**, which allows R package documentation to be written in comments. The concept of

structured, interwoven code and documentation has existed for many years, but existing systems that implement this for the R programming language do not tightly integrate with R code, leading to several drawbacks. This chapter attempts to address these issues and presents 2 contributions for documentation generation for the R community. First, we propose a new syntax for inline documentation of R code within comments adjacent to the relevant code, which allows for highly readable and maintainable code and documentation. Second, we propose an extensible system for parsing these comments, which allows the syntax to be easily augmented.

- In Chapter 9, we present named capture regular expressions, which can be used for data pre-processing. We explain the implementation of capturing subpattern locations and names, which are useful for extracting information from non-standard data files. As a result of this work, these features are available from the `regexpr` and `gregexpr` functions, in every copy of R starting with version 2.14.

1.3 Review of biology, genetics, neuroblastoma

This thesis discusses new methods in statistical machine learning with an emphasis on applications in bioinformatics. In this section, we review some concepts from biology that will help to understand the material in this thesis.

Review of genetics

Deoxyribonucleic acid (DNA) is the inherited molecule that encodes information about how biological cells and organisms function. DNA is organized into chromosomes in the nucleus of most eukaryotic cells. In normal human cells there are 23 pairs of chromosomes, as shown on the left of Figure 1.1. One chromosome of each pair comes from the father, and the other comes from the mother. One pair is called the sex chromosomes, which can be either a large X chromosome or a small Y chromosome. A female inherits an X chromosome from both parents, but a male inherits a Y from his father and an X from his mother. The other 22 pairs of chromosomes are called autosomes, and are numbered by roughly decreasing size from 1 to 22. As can be seen in Figure 1.1 and in more detail on the UCSC genome browser [Kent et al., 2002], chromosome 1 is the largest, chromosome 7 is roughly the same size as the X chromosome, and the Y chromosome is even smaller than chromosome 22.

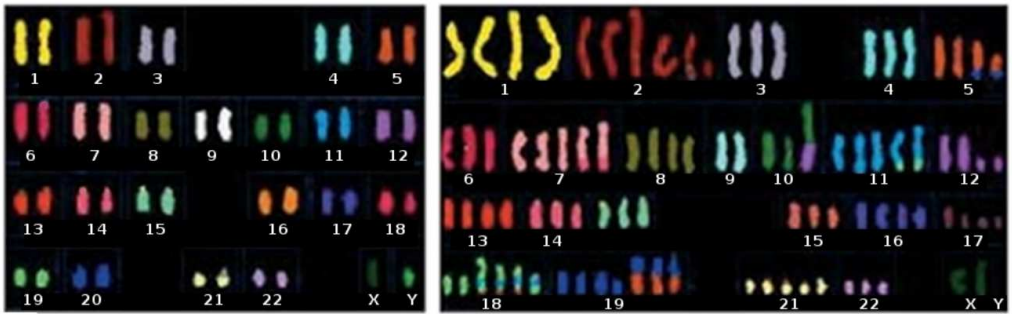

Figure 1.1: Spectral karyotype of copy number alterations in a cancer cell, described by Alberts et al. [2002]. In this assay, chromosomes are each stained with a specific color so they can be counted easily. **Left**: a normal genome with 2 copies of each autosome. **Right**: a cancer cell genome, which contains many abnormal duplications and translocations.

In this thesis, we will analyze array comparative genomic hybridization (aCGH) experiments that measure the copy number of cell samples. The goal of these experiments is to quantify the number of copies of each chromosome. As shown on the right of Figure 1.1, chromosomal copy number in cancer cells can deviate significantly from the normal level of 2 copies. A main contribution of this thesis are new methods that more accurately estimate chromosomal copy number from aCGH experiments.

An aCGH experiment simultaneously measures the copy number of a sample at each location in the human genome using microarray technology. To understand microarrays, one must consider how they are constructed based on the human genome sequence. The International Human Genome Sequencing Consortium noted that the published genome "...facilitates experimental tools to recognize cellular components—for example, detectors for mRNAs based on specific oligonucleotide probes...with confidence that these features provide a unique signature" [IHGSC, 2004]. Indeed, an aCGH microarray consists of many DNA probes, each which maps to a unique region of the genome. To analyze a tumor with aCGH, a tumor DNA sample is tagged with a fluorescent marker and then hybridized to the probes on the microarray. The level of fluorescence is measured for each probe, and the amount of fluorescence depends on the number of matching sequences in the tumor sample [Pinkel et al., 1998]. Thus, aCGH experiments yield measurements of copy number for as many locations in the genome as there are probes on the microarray.

Microarray pre-processing and normalization techniques are essential to aCGH data analysis, and a review of these techniques is given by Neuvial et al. [2010]. Data normalization procedures output a "logratio" signal that is roughly proportional to copy number [Pinkel et al., 1998]. It is defined as $\log_2(E/R)$, or the fold-change of the experimental signal E with respect to the reference sample R. This thesis analyzes these logratio signals and presents new mathematical models for predicting changes in copy number.

In Figure 1.2, we examine one tumor and show its "copy number profile," which is the plot of logratio measurements along the genome. The data come from the neuroblastoma data set, a database of copy number profiles that we have made public as part of this research. There are several copy number changes visible in Figure 1.2, and we present a new approach to detecting these changes in Chapter 3 of this thesis.

The terminology we will use when discussing copy number deduced from aCGH experiments is as follows:

- **Normal** level corresponds to logratio 0 and in non-cancerous tissues this corresponds to 2 copies.
- **Gains** are regions or entire chromosomes that are present in greater quantity than normal. Gains can result from entire chromosome duplications, or chromosomal translocations where just a region of one chromosome is copied and attached somewhere else in the genome.
- **Losses** are regions or entire chromosomes that are missing. These can result from abnormal events in cell division when the DNA copying machinery does not work properly.

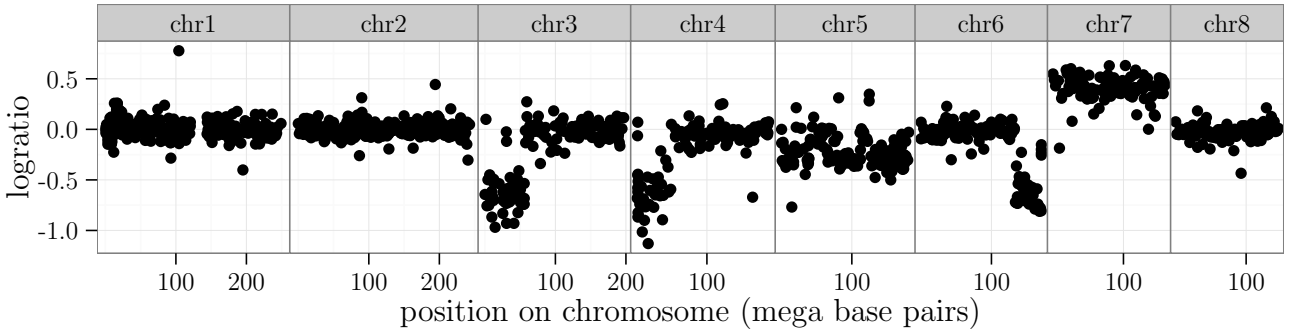

Figure 1.2: The copy number profile of a neuroblastoma tumor. An aCGH experiment was used to measure copy number of a tumor sample, and the normalized logratio measurements are plotted against chromosomal position. In this tumor, chromosomes 1, 2, and 8 exhibit a logratio of 0 which corresponds to the normal level of 2 copies. Chromosomes 3, 4, and 6 clearly show regions with a loss, and chromosome 7 exhibits a gain. Chromosomes 9-22 and X and Y were assayed but are not shown in this plot.

Review of neuroblastoma

We will focus on the analysis of neuroblastoma tumors taken from patients at diagnosis at the Institut Curie. Neuroblastoma is a pediatric cancer that manifests in infants and children, most frequently diagnosed at the age of 1 or 2 years. It affects approximately 1 child in 100,000, but its causes are still largely unknown [Maris, 2010]. One goal of this thesis is to develop statistical methods that more accurately characterize the genetic profiles of these tumors, so we may one day understand the genetic causes of this cancer.

Mysteriously, neuroblastoma in many children spontaneously disappears without treatment. So if we could use a genetic test to determine which children would have spontaneous remission, we could avoid treating those children with painful anti-cancer drugs and still be sure that they would recover.

There have been several previous attempts to characterize the genes involved in neuroblastoma development. The first gene observed in neuroblastoma to be amplified in many more copies than normal was N-myc, which is localized on the short arm of chromosome 2 [Schwab et al., 1984]. More recently, mutations and copy number changes in the PHOX2B and ALK genes have been observed in association with neuroblastoma [Janoueix-Lerosey et al., 2008, Trochet et al., 2004]. So there are several genes known to be associated to neuroblastoma, and we may be able to identify new genes by examining copy number profiles of tumors.

Clinical outcome of patients with neuroblastoma has been shown to be worse for tumors with segmental alterations or breakpoints in specific genomic regions [Janoueix-Lerosey et al., 2009, Schleiermacher et al., 2010]. In those studies, copy number of large chromosomal regions was estimated by visual inspection of the copy number profiles. In this thesis, we develop mathematical models that give more precise estimation of the copy number alterations, yielding much smaller regions where we may find genes associated with neuroblastoma.

1.4 Review of convex relaxation, optimality, and algorithms

In this thesis, we will make extensive use of convex optimization to formulate and solve statistical learning problems. A good introduction to statistical learning is given by Hastie et al. [2009]. We use the convex optimization notation conventions of Boyd and Vandenberghe [2004].

Convex relaxation renders difficult problems tractable

We will be concerned with constrained optimization problems with real vectors, which generally take the following form.

- Let $x = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}'$ be the vector of n optimization variables.
- Let $f_0 : \mathbb{R}^n \to \mathbb{R}$ be the objective function.
- For every inequality constraint $i \in \{1, \ldots, m\}$, let $f_i : \mathbb{R}^n \to \mathbb{R}$ be an inequality constraint function.
- For every equality constraint $i \in \{1, \ldots, p\}$, let $h_i : \mathbb{R}^n \to \mathbb{R}$ be the equality constraint function i.

The optimization problem is to find a vector of variables $x \in \mathbb{R}^n$ such that the objective is minimal.

$$\begin{array}{ll} \text{min} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \text{ for all } i \in \{1, \ldots, m\} \\ & h_i(x) = 0 \text{ for all } i \in \{1, \ldots, p\}. \end{array} \tag{1.1}$$

In other words, we would like to find an optimal vector x^* such that each constraint $f_i(x^*) \leq 0$ and $h_i(x^*) = 0$ is satisfied, and for any x that also satisfies the constraints we have $f_0(x) \geq f_0(x^*)$. Note that there can be several $x_1 \neq x_2$ which attain the minimal value $f^* = f_0(x_1) = f_0(x_2)$, as shown in the top of Figure 1.3.

Many real-world statistical problems can be written in the form of (1.1), and in this thesis we will often introduce problems of this form. However, real-world problems are often intractable due to lack of structure.

A classical intractable problem from statistics is the best subset variable selection problem in regression. Given a matrix of inputs $A \in \mathbb{R}^{l \times n}$ and a

vector of outputs $y \in \mathbb{R}^l$, we would like to select the best $k < n$ variables:

$$\begin{aligned} \min \quad & f_0(x) = ||Ax - y||_2^2 \\ \text{subject to} \quad & f_1(x) = ||x||_0 = \sum_{i=1}^{n} 1_{x_i \neq 0} \leq k \end{aligned} \tag{1.2}$$

In this problem, there are $p = 0$ equality constraints and there is $m = 1$ inequality constraint function f_1. It is defined as the ℓ_0 pseudo-norm of the vector x, which is a sum of indicator functions that count if the entries are non-zero.

The constraint function f_1 is neither differentiable nor continuous, and this lack of structure can be seen in the fourth panel of Figure 1.3. So solving this problem requires calculating the least squares solution and checking the value of f_0 for $\binom{n}{k} = (n!)/(k!(n-k)!)$ combinations of variables. Due to the factorials, with large n this quantity becomes too large to handle using even the fastest computers. Technically, we say that this problem is NP-hard [Cormen et al., 1990, Chapter 34].

Using the technique of convex relaxation, we can alter this problem and create a new one which is tractable. In the case of best subset selection for regression, we can alter the constraint function f_1. Instead of constraining the number of nonzero entries of the vector x, we can constrain the size of x using the ℓ_1 norm:

$$\begin{aligned} \min \quad & ||Ax - y||_2^2 \\ \text{subject to} \quad & ||x||_1 = \sum_{i=1}^{n} |x_i| \leq s, \end{aligned} \tag{1.3}$$

where $s \in \mathbb{R}^+$ is the maximum size of the parameter vector. This alteration results in a modified regression problem commonly known as the Lasso [Tibshirani, 1996]. Like the original problem (1.2), the Lasso problem (1.3) has $p = 0$ equality constraints and $m = 1$ inequality constraint. The ℓ_1 norm $||x||_1$ is a function with a specific structure, as shown in the middle panel of Figure 1.3. In particular, it is a convex function, which we define precisely in the next section. We can exploit the convexity and piecewise linearity of the ℓ_1 norm to solve problem 1.3 efficiently.

One could worry that relaxing the ℓ_0 pseudo-norm to the ℓ_1 norm significantly changes the problem, so that we will not get a good solution to the initial problem. However, recent results show that with just a few more observations, the relaxed problem is able to recover the same solution [Candès and Tao, 2009].

In summary, the technique of convex relaxation can be used on intractable non-convex optimization problems. The idea is to replace a non-convex function with a convex approximation. In particular, we often replace the ℓ_0 pseudo-norm with the ℓ_1 norm. The result is a modified optimization problem which is convex and thus can be solved efficiently. In the next section, we discuss several methods for solving convex optimization problems.

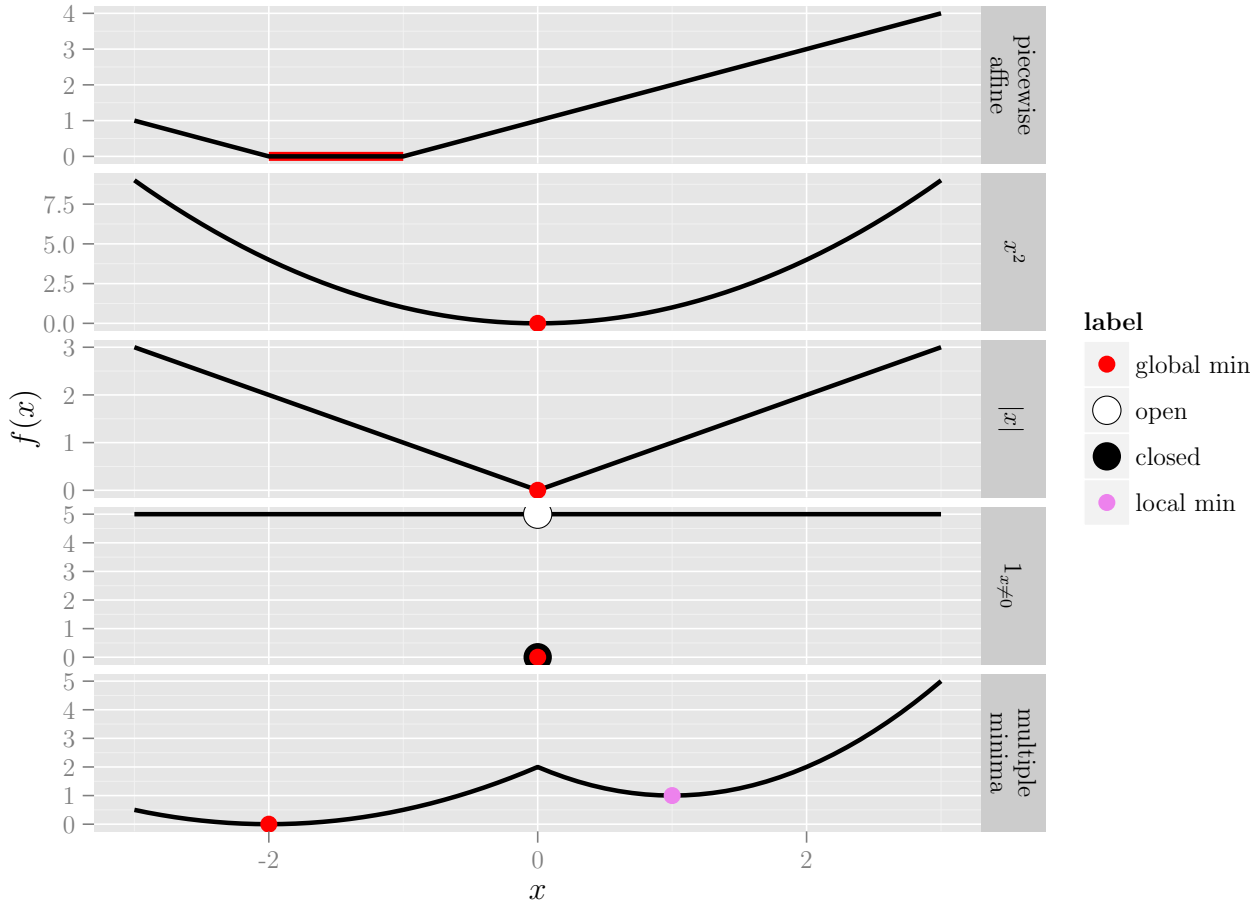

Figure 1.3: Some functions $f : \mathbb{R} \to \mathbb{R}$ (black) and their global minima (red). From top to bottom: a convex non-smooth piecewise affine function with many global minima; the squared ℓ_2 norm is a convex smooth function with a unique global minimum; the ℓ_1 norm is a convex non-smooth piecewise linear function with a unique global minimum; the ℓ_0 pseudo-norm is a discontinuous non-convex function with a unique global minimum; a continuous non-convex function with 2 local minima, one of which is global.

Optimality conditions for convex functions

In the previous section we discussed the technique of convex relaxation, which replaces non-convex functions with convex approximations. To be precise, a function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for all $x, y \in \mathbb{R}^n$, and for all θ such that $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y). \tag{1.4}$$

To understand the definition of convexity, consider the examples in Figure 1.3. Geometrically, the definition of convexity means that you can choose any two points x, y on the plot of f, and the line segment that connects x and y will never underestimate the function f.

When the objective function f_0 is convex and the constraint functions $f_1, \ldots, f_m$ are convex, then we have a convex optimization problem. Convex optimization problems are tractable since we can analyze them and use their gradients or subgradients to solve them [Boyd and Vandenberghe, 2004].

If a function $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, the gradient $\nabla f(x) \in \mathbb{R}^n$ at the point $x \in \mathbb{R}^n$ is the vector of partial derivatives of that function:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \ldots & \frac{\partial f(x)}{\partial x_n} \end{bmatrix}'. \tag{1.5}$$

The gradient is useful since it allows us to characterize the stationary points of f. In particular, we know that any point x such that $\nabla f(x) = 0$ is either a local minimum, local maximum, or inflection point of f.

However, functions may have several stationary points, as shown in the bottom panel of Figure 1.3. For a function f with several local minima, there are several points x each which satisfy $\nabla f(x) = 0$. So for general functions f, the gradient condition $\nabla f(x) = 0$ is not sufficient to ensure that x is the global minimum.

In contrast, we can use the gradient condition to find the global minimum of differentiable convex functions. In particular, for every differentiable convex function f that achieves its infimum f^*, we have

$$f(x) = f^* \Leftrightarrow \nabla f(x) = 0. \tag{1.6}$$

For example, the squared norm x^2 in the second panel of Figure 1.3 is a convex differentiable function whose global minimum is characterized by the gradient condition. So we can characterize the global minimum of any smooth convex function f using the gradient condition.

However, we would also like to analyze convex, non-differentiable functions such as the ℓ_1 norm and the piecewise affine function shown in Figure 1.3. To analyze these functions, we define the subdifferential for any convex function $f : \mathbb{R}^n \to \mathbb{R}$ as

$$\partial f(x) = \{z \in \mathbb{R}^n \mid f(x) - f(y) \geq z'(x - y) \text{ for any } y \in \mathbb{R}^n\}. \tag{1.7}$$

In general, the subdifferential contains all vectors $z \in \mathbb{R}^n$ that define tangent planes of the convex function f at x. For smooth functions, the subdifferential contains just the gradient vector $\partial f(x) = \{\nabla f(x)\}$. We plot a non-differentiable function with its subdifferential in Figure 1.4.

We will use subdifferentials to characterize the optima of non-smooth convex functions. In particular, we use the fact that if the subdifferential

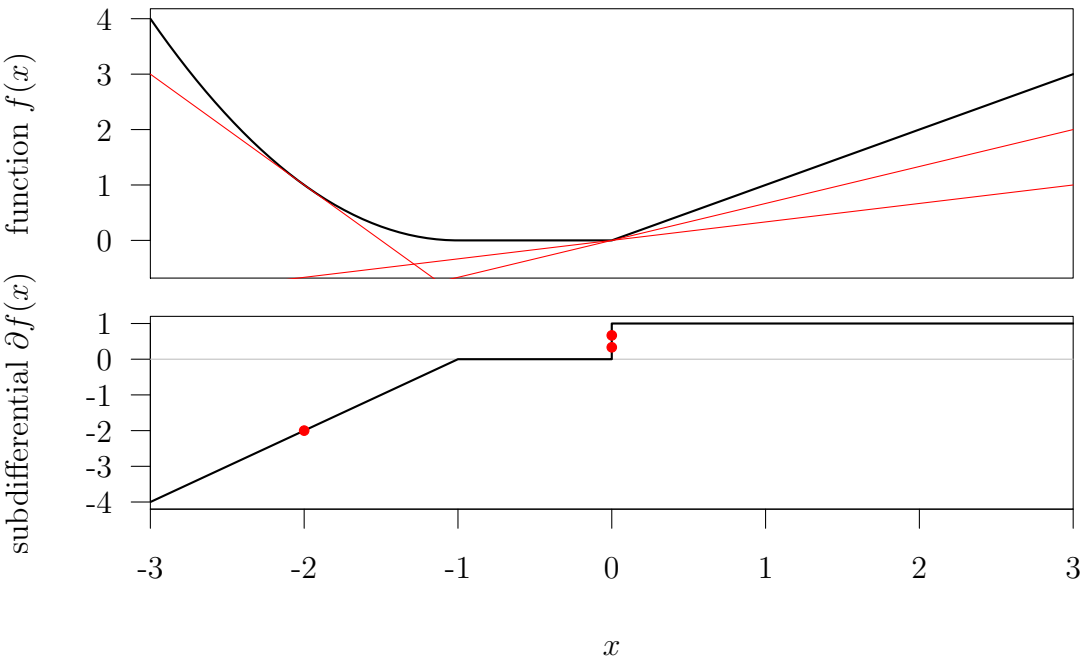

Figure 1.4: **Top**: a convex function $f : \mathbb{R} \to \mathbb{R}$. **Bottom**: its subdifferential $\partial f : \mathbb{R} \to 2^{\mathbb{R}}$. At every $x \neq 0$, the function $f(x)$ is differentiable so $\partial f(x) = \{\nabla f(x)\}$. At the non-differentiable point we have $\partial f(0) = [0, 1]$. The subdifferential at x gives the slope (red dots) of tangent lines (red lines) of $f(x)$. Note that since f is convex, the condition $0 \in \partial f(x)$ characterizes its global minima.

at x contains the zero vector $0 \in \partial f(x)$, then x is a minimum of f:

$$f(x) = f^* \Leftrightarrow 0 \in \partial f(x). \tag{1.8}$$

Note that when f is smooth this reduces to the usual gradient optimality condition $\nabla f(x) = 0$.

However, the minimum of a non-smooth convex function may not be unique. As shown in the top panel of Figure 1.3, there could be several distinct optimal $x_1 \neq x_2$ with the same optimal value $f(x_1) = f(x_2) = f^*$. It is sufficient to find any one of these optimal vectors.

Algorithms for convex optimization

There are several practical methods or "solvers" that one can use to solve convex optimization problems. Let us take the point of view of a statistical programmer who will write the code that solves problem (1.3) for a particular set of inputs A and outputs y. There is a tradeoff between the

- time it takes to analyze the problem structure on paper and write the code that implements the solver, and the
- computational running time and memory requirements.

This thesis will focus on solvers that take a while to implement, but run very fast on the computer. For example, we can solve problem (1.3) using FISTA, a Fast Iterative Shrinkage-Thresholding Algorithm [Beck and Teboulle, 2009]. To use FISTA, we must analyze the structure of the optimization problem. First we must take the equivalent Lagrange formulation of the problem, which results in the following unconstrained problem

$$\min_x ||Ax - y||_2^2 + \lambda ||x||_1. \tag{1.9}$$

The Langrange multiplier $\lambda \in \mathbb{R}^+$ is a fixed parameter that controls the size of the parameter vector x, and plays the same role as s in problem (1.3). To use FISTA to solve this problem, we must find the gradient and a Lipshitz constant of the smooth part $||Ax-y||_2^2$. Then, we must derive the proximal operator of the non-smooth part $||x||_1$. Finally, we must derive a stopping condition from the subdifferential condition (1.8) that determines when we have achieved an optimal solution vector x. When all these steps have been done on paper, we can write the code that implements FISTA, and solves problem (1.9) very efficiently.

Active-set methods are another class of computationally efficient optimization algorithms that we will exploit in this thesis. The idea in active-set

methods is to first determine which subset of variables x_i are active, and then save time by only considering those variables in the optimization. For example, we know that in problem (1.9) some variables x_i will be exactly zero in the optimal solution. We refer to the other non-zero variables as the active set. Implementing an active-set method requires detailed analysis of the optimality conditions to determine which variables are active. In Chapter 2 of this thesis, we will use active-set methods to calculate the clusterpath.

A special class of active-set methods that we will discuss in this thesis are homotopy methods. For example, the solutions to problem (1.9) can be found for all $\lambda \in \mathbb{R}^+$ using a homotopy method called the Least Angle Regression algorithm (LARS) [Efron et al., 2004]. We refer to the set of optimal x as λ changes as the "path of solutions," which is why homotopy methods are often called path-following algorithms. These methods are useful in statistical learning applications since we often want to use a regularized model with an intermediate value of λ, but we do not know what specific value of λ to use in advance. So in practice we use cross-validation to fit the entire path of solutions on a training set of data, and pick the model complexity λ for which prediction error is minimal on a test set of data. A homotopy algorithm can be applied whenever the optimization problem contains a piecewise quadratic term and a piecewise linear term [Rosset and Zhu, 2007]. Implementing homotopy methods is usually quite time-consuming, and requires analysis of how the optimality condition changes with λ. In this thesis, we use a homotopy method in Chapter 2 to calculate the ℓ_1 clusterpath.

Another way to solve problem (1.3) is to use a generic quadratic programming (QP) solver such as the **quadprog** package in R [Turlach and Weingessel, 2011]. Using this method is significantly simpler to implement than the methods discussed above. In particular, we do not need to derive the gradient, Lipshitz constant, proximal operator, optimality conditions, or stopping condition. We just need to convert the problem to standard form:

$$\begin{aligned} \min_{x\in\mathbb{R}^n, x^+\in\mathbb{R}^n, x^-\in\mathbb{R}^n} \quad & \frac{1}{2}x'A'Ax - y'Ax \\ \text{subject to} \quad & \forall i \in \{1,\dots,n\},\ x_i = x_i^+ - x_i^- \\ & \forall i \in \{1,\dots,n\},\ x_i^+ \geq 0 \\ & \forall i \in \{1,\dots,n\},\ x_i^- \geq 0 \\ & \sum_{i=1}^n x_i^+ + x_i^- \leq s, \end{aligned} \tag{1.10}$$

where x_i^+, x_i^- are variables that we introduce so we can write the constraint on the ℓ_1 norm in standard form. Then, we apply the QP solver, which implements the method of Goldfarb and Idnani [1983]. Though using a QP solver requires much less derivation on paper, it generally requires more time for the computer to calculate the solution. Conceptually, since the QP solver uses just the objective function and constraints in standard form, it is unable to find the solution as fast as other methods such as FISTA that exploit the problem's structure.

Finally, one of the fastest ways to write the code that solves (1.3) is using disciplined convex programming [Grant et al., 2006]. Using this technique, a library such as `cvxmod` is used to write code that describes the optimization problem [Mattingley and Boyd, 2008]. Then, the `cvxmod` library analyzes the program, translates it into standard form, and applies a standard solver. So this is even faster to code than using a QP solver, since we do not even need to write the problem in standard form. In contrast, these methods often run very slowly on the computer, and so are only feasible for optimization problems with few variables and constraints.

So in practice there are a variety of different methods that can be used to find the solution to statistical learning problems such as the Lasso (1.3). Each method has its own strengths and weaknesses in terms of coding and calculation time. However, these methods can be used to complement one another. In this thesis, we will introduce new algorithms that take a while to code, but result in very fast calculation times. We will use the computationally slower, more generic solvers to check that our solvers find the correct solutions.

1.5 Review of current statistical software

In part 1 of this thesis, we will focus on defining and characterizing new machine learning algorithms for data analysis. To use them in practical analyses of real data, these algorithms must be implemented in computers by writing software.

Software for data analysis can be written in many languages, but for this thesis I used the R [R Development Core Team, 2012], Python [Pilgrim, 2004], C [Kernighan and Ritchie, 1988], and C++ programming languages [Stroustrup, 1997]. These languages are good choices for scientific computing for 2 main reasons.

- The compiler or interpreter for each of these languages is available free of charge on the internet. This permits anyone to download the code that implements my algorithms and run it on their computer. This is particularly important for reproducible research, so that my colleagues in the international research community may easily verify the results that I report.

- The compiler or interpreter for each language has open source code. This is particularly important for science so we can read the source code to see exactly how everything works. Also, this means that we can alter the base language interpreter if necessary. I show one example of this in Chapter 9 when I show how to add named capture regular expressions to R.

In part 2 of this thesis, I discuss statistical software contributions that I have implemented in R, a language and environment for statistical computing and graphics. R has data structures, built-in functions, and an interactive interface that make it particularly well-suited for data analysis. R is collaboratively developed by volunteers from all over the world, and this thesis presents three contributions to that effort. Although I do not discuss detailed usage of the R language in this thesis, I refer the reader to the introduction given by Murrell [2009, Chapter 9].

The software I wrote in support of part 2 of this thesis works only with R, but the underlying ideas and algorithms can be implemented using any programming language. So I give detailed explanations that should permit implementation of these ideas in other languages.

Part I

Segmentation and clustering algorithms, with applications to bioinformatics

Chapter 2

Clusterpath: a hierarchical clustering algorithm using convex fusion penalties

Some content of this chapter comes from the Clusterpath peer-reviewed conference paper that was published in the proceedings of the ICML 2011 [Hocking et al., 2011]. This is joint work with Armand Joulin and my advisors Francis Bach and Jean-Philippe Vert.

Chapter summary

We present a new clustering algorithm by proposing a convex relaxation of hierarchical clustering, which results in a family of objective functions with a natural geometric interpretation. We give efficient algorithms for calculating the continuous regularization path of solutions, and discuss relative advantages of the parameters. Our method experimentally gives state-of-the-art results similar to spectral clustering for non-convex clusters, and has the added benefit of learning a tree structure from the data.

2.1 Introduction

In the analysis of multivariate data, cluster analysis is a family of unsupervised learning techniques that allows identification of homogenous subsets of data. Algorithms such as k-means, Gaussian mixture models, hierarchical clustering, and spectral clustering allow recognition of a variety of cluster shapes. However, all of these methods suffer from instabilities, either because they are cast as non-convex optimization problems, or because they

rely on hard thresholding of distances. Several convex clustering methods have been proposed, but some only focus on the 2-class problem [Xu et al., 2004], and others require arbitrary fixing of minimal cluster sizes in advance [Bach and Harchoui, 2008]. The main contribution of this work is the development of a new convex hierarchical clustering algorithm that attempts to address these concerns.

In recent years, sparsity-inducing norms have emerged as flexible tools that allow variable selection in penalized linear models. The Lasso and group Lasso are now well-known models that enforce sparsity or group-wise sparsity in the estimated coefficients [Tibshirani, 1996, Yuan and Lin, 2006]. Another example, more useful for clustering, is the fused Lasso signal approximator (FLSA), which has been used for segmentation and image denoising [Tibshirani and Saunders, 2005]. Furthermore, several recent papers have proposed optimization algorithms for linear models using ℓ_1 [Chen et al., 2010, Shen and Huang, 2010] and ℓ_2 [Vert and Bleakley, 2010] fusion penalties. There has even been some previous work on using ℓ_1 fusion penalties for the clustering problem [Lindsten et al., 2011, Pelckmans et al., 2005], but in this chapter we introduce the ℓ_2 and ℓ_∞ norms for clustering. This chapter develops a family of fusion penalties that results in the "clusterpath," a hierarchical regularization path useful for clustering problems.

Motivation by relaxing hierarchical clustering

Hierarchical or agglomerative clustering is calculated using a greedy algorithm, which for n points in $\mathbb{R}^p$ recursively joins the points which are closest together. For the data matrix $X \in \mathbb{R}^{n\times p}$ this suggests the problem

$$\begin{aligned} \min_{\alpha\in\mathbb{R}^{n\times p}} \quad & \frac{1}{2}||\alpha - X||_F^2 \\ \text{subject to} \quad & \sum_{i<j} 1_{\alpha_i\neq\alpha_j} \leq t, \end{aligned} \tag{2.1}$$

where $||\cdot||_F^2$ is the squared Frobenius norm, $\alpha_i \in \mathbb{R}^p$ is row i of α, and $1_{\alpha_i\neq\alpha_j}$ is 1 if $\alpha_i \neq \alpha_j$, and 0 otherwise. We use the notation $\sum_{i<j} = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}$ to sum over all the $n(n-1)/2$ pairs of data points. Note that when we fix $t \geq n(n-1)/2$ the problem is unconstrained and the solutions are $\alpha_i = X_i$ for all i. If $t = n(n-1)/2 - 1$, we force one pair of coefficients to fuse, and this is equivalent to the first step in hierarchical clustering. When $t = 0$ at the other end of the path, all optimal coefficients α_i are equal so the solution is $\alpha_i = \bar{X} = \sum_{i=1}^{n} X_i/n$ for all i. However, in general this is a difficult combinatorial optimization problem.

Instead of tackling (2.1) directly, we propose a convex relaxation. This results in the family of optimization problems defined by

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^{n \times p}} \quad & \frac{1}{2}||\alpha - X||_F^2 \\ \text{subject to} \quad & \Omega_q(\alpha) = \sum_{i<j} w_{ij}||\alpha_i - \alpha_j||_q \leq t, \end{aligned} \tag{2.2}$$

where $w_{ij} > 0$, and $||\cdot||_q$, $q \in \{1, 2, \infty\}$ is the ℓ_q-norm on $\mathbb{R}^p$, which will induce sparsity in the differences of the rows of α. When rows fuse we say they form a cluster, and the continuous regularization path of optimal solutions formed by varying t is what we call the "clusterpath."

This optimization problem has an equivalent geometric interpretation (Figure 2.1). For the identity weights $w_{ij} = 1$, the solution corresponds to the closest points α to the points X, subject to a constraint on the sum of distances between pairs of points. For general weights, we constrain the total area of the rectangles of width w_{ij} between pairs of points.

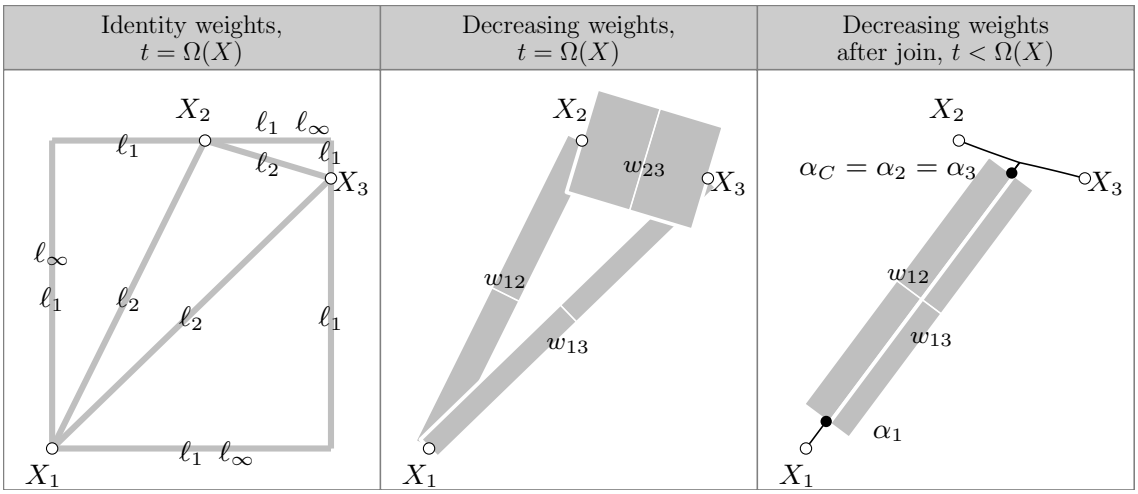

Figure 2.1: Geometric interpretation of the optimization problem (2.2) for data $X \in \mathbb{R}^{3\times 2}$. **Left:** with the identity weights $w_{ij} = 1$, the constraint $\Omega_q(\alpha) = \sum_{i<j} w_{ij}||\alpha_i - \alpha_j||_q \leq t$ is the ℓ_q distance between all pairs of points, shown as grey lines. **Middle:** with general weights w_{ij}, the ℓ_2 constraint is the total area of rectangles between pairs of points. **Right:** after constraining the solution, α_2 and α_3 fuse to form the cluster C, and the weights are additive: $w_{1C} = w_{12} + w_{13}$.

This parameterization in terms of t is cumbersome when comparing data sets X since we take $0 \leq t \leq \Omega_q(X)$, so we introduce the following parametrization with $0 \leq s \leq 1$:

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^{n \times p}} \quad & \frac{1}{2}||\alpha - X||_F^2 \\ \text{subject to} \quad & \Omega_q(\alpha)/\Omega_q(X) \leq s. \end{aligned} \tag{2.3}$$

The equivalent Langrangian dual formulation will also be convenient for optimization algorithms:

$$\min_{\alpha \in \mathbb{R}^{n \times p}} f_q(\alpha, X) = \frac{1}{2}||\alpha - X||_F^2 + \lambda \Omega_q(\alpha). \tag{2.4}$$

The above optimization problems require the choice of predefined, pair-specific weights $w_{ij} > 0$, which can be used to control the geometry of the solution path. In most of our experiments we use weights that decay with the distance between points $w_{ij} = \exp(-\gamma||X_i - X_j||_2^2)$, which results in a clusterpath that is sensitive to local density in the data. Another choice for the weights is $w_{ij} = 1$, which allows efficient computation of the ℓ_1 clusterpath, as will be shown in Section 2.2.

Visualizing the geometry of the clusterpath

In this work we develop dedicated algorithms for solving the clusterpath which allow scaling to large data, but initially we used `cvxmod` for small problems [Mattingley and Boyd, 2008], as the authors do in a similar formulation [Lindsten et al., 2011].

We used `cvxmod` to compare the geometry of the clusterpath for several choices of norms and weights (Figure 2.2). Note the piecewise linearity of the ℓ_1 and ℓ_∞ clusterpath, which can be exploited to find the solutions using efficient path-following homotopy algorithms. Furthermore, it is evident that the ℓ_2 path is invariant to rotation of the input data X, whereas the others are not.

The rest of this chapter is organized as follows. In Section 2.2, we propose a specific method for each norm for solving the problem. In Section 2.3, we propose an extension of our methods to spectral representations, thus providing a convex formulation of spectral clustering. Finally, in Section 3.3 we empirically compare the clusterpath to standard clustering methods.

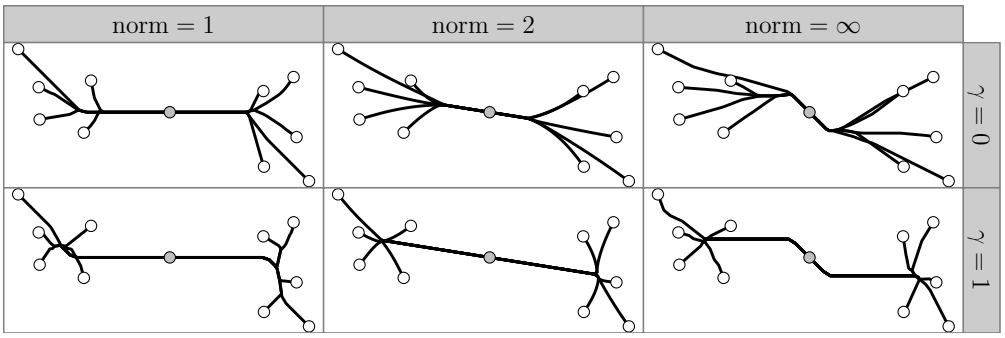

Figure 2.2: Some random normal data $X \in \mathbb{R}^{10\times 2}$ were generated (white dots) and their mean $\bar{X}$ is marked in grey in the center. The clusterpath (black lines) was solved using `cvxmod` for 3 norms (panels from left to right) and 2 weights (panels from top to bottom), which were calculated using $w_{ij} = \exp(-\gamma||X_i - X_j||^2)$. For $\gamma = 0$, we have $w_{ij} = 1$.

2.2 Optimization

A homotopy algorithm for the ℓ_1 solutions

For the problem involving the ℓ_1 penalty, we first note that the problem is separable on dimensions. The cost function can be written as

$$\begin{aligned} f_1(\alpha, X) &= \frac{1}{2}||\alpha - X||_F^2 + \lambda\Omega_1(\alpha) && (2.5)\\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{p}(\alpha_{ik} - X_{ik})^2 + \lambda\sum_{i<j} w_{ij}\sum_{k=1}^{p}|\alpha_{ik} - \alpha_{jk}| && (2.6)\\ &= \sum_{k=1}^{p}\left[\frac{1}{2}\sum_{i=1}^{n}(\alpha_{ik} - X_{ik})^2 + \lambda\sum_{i<j} w_{ij}|\alpha_{ik} - \alpha_{jk}|\right] && (2.7)\\ &= \sum_{k=1}^{p} f_1(\alpha^k, X^k), && (2.8) \end{aligned}$$

where $\alpha^k \in \mathbb{R}^n$ is the k-th column from α. Thus, solving the minimization with respect to the entire matrix X just amounts to solving p separate minimization subproblems:

$$\min_{\alpha\in\mathbb{R}^{n\times p}} f_1(\alpha, X) = \sum_{k=1}^{p}\min_{\alpha^k\in\mathbb{R}^n} f_1(\alpha^k, X^k). \quad (2.9)$$

For each of these subproblems, we can exploit the FLSA path algorithm [Hoefling, 2009]. This is a homotopy algorithm similar to the LARS that exploits the piecewise linearity of the path to very quickly calculate the entire set of solutions [Efron et al., 2004].

In the LARS, variables jump in and out the active set, and we must check for these events at each step in the path. The analog in the FLSA path algorithm is the necessity to check for cluster splits, which occur when the optimal solution path requires unfusing a pair coefficients. Cluster splits were not often observed on our experiments, but are also possible for the ℓ_2 clusterpath, as illustrated in Figure 2.4. The FLSA path algorithm checks for a split of a cluster of size n_C by solving a max-flow problem using a push-relabel algorithm, which has complexity $O(n_C^3)$ [Cormen et al., 2001]. For large data sets, this can be prohibitive, and for any clustering algorithm, splits make little sense.

One way around this bottleneck is to choose weights w in a way such that no cluster splits are possible in the path. The modified algorithm then only considers cluster joins, and results in a complexity of $O(n \log n)$ for a

single dimension, or $O(pn \log n)$ for p dimensions. One choice of weights that results in no cluster splits is the identity weights $w_{ij} = 1$, which we prove below.

The ℓ_1 clusterpath using $w_{ij} = 1$ contains no splits

The proof will establish a contradiction by examining the necessary conditions on the optimal solutions during a cluster split. We will need the following lemma.

Lemma 1. *Let $C = \{i : \alpha_i = \alpha_C\} \subseteq \{1, ..., n\}$ be the cluster formed after the fusion of all points in C, and let $w_{jC} = \sum_{i \in C} w_{ij}$. At any point in the regularization path, the slope of its coefficient is given by*

$$v_C = \frac{d\alpha_C}{d\lambda} = \frac{1}{|C|} \sum_{j \notin C} w_{jC} \operatorname{sign}(\alpha_j - \alpha_C). \tag{2.10}$$

Proof. Consider the following sufficient optimality condition, for all $i = 1, \ldots, n$:

$$0 = \alpha_i - X_i + \lambda \sum_{\substack{j \neq i \\ \alpha_i \neq \alpha_j}} w_{ij} \operatorname{sign}(\alpha_i - \alpha_j) + \lambda \sum_{\substack{j \neq i \\ \alpha_i = \alpha_j}} w_{ij} \beta_{ij}, \tag{2.11}$$

with $|\beta_{ij}| \leq 1$ and $\beta_{ij} = -\beta_{ji}$ [Hoefling, 2009]. We can rewrite the optimality condition for all $i \in C$:

$$0 = \alpha_C - X_i + \lambda \sum_{j \notin C} w_{ij} \operatorname{sign}(\alpha_C - \alpha_j) + \lambda \sum_{i \neq j \in C} w_{ij} \beta_{ij}. \tag{2.12}$$

Furthermore, by summing each of these equations, we obtain the following:

$$\alpha_C = \bar{X}_C + \frac{\lambda}{|C|} \sum_{j \notin C} w_{jC} \operatorname{sign}(\alpha_j - \alpha_C), \tag{2.13}$$

where $\bar{X}_C = \sum_{i \in C} X_i / |C|$. Taking the derivative with respect to λ gives us the slope v_C of the coefficient line for cluster C, proving Lemma 1. □

We will use Lemma 1 to prove by contradiction that cluster splitting is impossible for the case $w_{ij} = 1$ for all i and j.

Theorem 1. *Taking $w_{ij} = 1$ for all i and j is sufficient to ensure that the ℓ_1 clusterpath contains no splits.*

Proof. Consider at some λ the optimal solution α, and let C be a cluster of any size among these optimal solutions. Denote the set $\overline{C} = \{i : \alpha_i > \alpha_C\}$ the set of indices of all larger optimal coefficients and $\underline{C} = \{i : \alpha_i < \alpha_C\}$ the set of indices of all smaller optimal coefficients. Note that $\overline{C} \cup \underline{C} \cup C = \{1, \ldots, n\}$.

Now, assume C splits into C_1 and C_2 such that $\alpha_1 > \alpha_2$. By Lemma 1, if this situation constitutes an optimal solution, then the slopes are:

$$\begin{aligned} v_{C_1} &= \frac{1}{|C_1|}\left(\sum_{j\in\overline{C}} w_{jC_1} - \sum_{j\in C_2} w_{jC_1} - \sum_{j\in\underline{C}} w_{jC_1}\right) \\ v_{C_2} &= \frac{1}{|C_2|}\left(\sum_{j\in\overline{C}} w_{jC_2} + \sum_{j\in C_1} w_{jC_2} - \sum_{j\in\underline{C}} w_{jC_2}\right). \end{aligned} \tag{2.14}$$

For the identity weights, this simplifies to

$$\begin{aligned} v_{C_1} &= |\overline{C}| - |C_2| - |\underline{C}| \\ v_{C_2} &= |\overline{C}| + |C_1| - |\underline{C}|. \end{aligned} \tag{2.15}$$

Thus $v_{C_1} < v_{C_2}$ which contradicts the assumption that $\alpha_1 > \alpha_2$, forcing us to conclude that no split is possible for the identity weights. □

Thus the simple FLSA algorithm of complexity $O(n \log n)$ without split checks is sufficient to calculate the ℓ_1 clusterpath for 1 dimension using the identity weights.

Furthermore, since the clusterpath is strictly agglomerative on each dimension, it is also strictly agglomerative when independently applied to each column of a matrix of data. Thus the ℓ_1 clusterpath for a matrix of data is also strictly agglomerative, and results in an algorithm of complexity $O(pn \log n)$. This is an interesting alternative to hierarchical clustering, which normally requires $O(pn^2)$ space and time for $p > 1$. Thus the ℓ_1 clusterpath can be used when n is very large, and hierarchical clustering is not feasible.

Implementation details

To calculate the ℓ_1 clusterpath in $O(n \log n)$ operations we use an algorithm that keeps track of information for each cluster and updates this information after every cluster fusion. First, sort X such that $X_1 < \cdots < X_n$, which can be done in $O(n \log n)$ operations on average using the quicksort algorithm [Cormen et al., 1990]. We start at $\lambda = 0$, where the optimal $\alpha_i = X_i$ for all points i. Thus at the beginning of the path we have n clusters $C_1, \ldots, C_n$, with $C_i = \{i\}$ for all i. By Lemma 1, we have cluster velocity

$$\begin{aligned} v_{C_i} &= \frac{1}{|C_i|} \sum_{j \notin C_i} w_{jC_i} \operatorname{sign}(\alpha_j - \alpha_{C_i}) && (2.16) \\ &= \sum_{j \neq i} w_{ij} \operatorname{sign}(\alpha_j - \alpha_i) && (2.17) \\ &= \sum_{j \neq i} \operatorname{sign}(X_j - X_i) && (2.18) \\ &= |\overline{C}_i| - |\underline{C}_i| && (2.19) \\ &= n - 1 - 2(i-1), && (2.20) \end{aligned}$$

for all points $i \in \{1, \ldots, n\}$ when we take the identity weights $w_{ij} = 1$. This closed-form expression allows calculation of the starting velocities in $O(n)$ operations.

For every cluster $i \in \{1, \ldots, n-1\}$, we can calculate the λ when it may fuse with cluster $i+1$ using the current v_i, λ_i, and α_i values. Each cluster $i \in \{1, \ldots, n\}$ follows this line:

$$\alpha_i - \alpha = v_i(\lambda_i - \lambda), \tag{2.21}$$

so it can be shown that line of each cluster $i \in \{1, \ldots, n-1\}$ will intersect with the line of cluster $i+1$ at

$$\lambda = \frac{\alpha_i - \alpha_{i+1} + v_i\lambda_i - v_{i+1}\lambda_{i+1}}{v_{i+1} - v_i}. \tag{2.22}$$

This closed form expression allows us to calculate the locations of the $n-1$ initial possible fusion events in $O(n)$ operations.

We keep track of fusion events in a red-black tree, as implemented in the multimap container in the C++ Standard Template Library [Cormen et al., 1990]. We insert the $n-1$ possible initial fusion events in the red-black tree, and each insert takes $O(\log n)$ operations. In fact we are only interested in the fusion event with the minimal λ value. The red-black tree keeps fusion events ordered by λ, so we can find the minimal λ fusion event in $O(1)$ operations.

We keep track of clusters using a linked list, as shown in Figure 2.3. When 2 clusters C_1 and C_2 fuse, we can use the following identity to get the velocity of the new cluster:

$$v_{\text{new}} = \frac{v_1|C_1| + v_2|C_2|}{|C_1| + |C_2|}. \tag{2.23}$$

Since following the links between clusters and events takes only $O(1)$ operations, the bottleneck is the addition of the 2 new possible fusion events, which takes $O(\log n)$ operations using the red-black tree.

Thus we perform $n - 1$ fusion events with $O(\log n)$ operations each, for a total of $O(n \log n)$ operations. The algorithm is finished when there is only one cluster remaining. It returns the pointer to the last cluster, which is in fact a tree that contains links to all the smaller clusters.

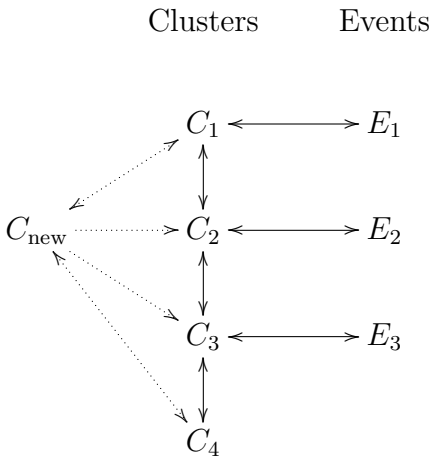

Figure 2.3: The data structure used to store clusters for efficient fusion events. The clusters are represented by a doubly-linked list, represented by the vertical arrows. Each cluster except the last is linked to an event corresponding to the merge with the cluster below. These events are kept in a red-black tree for $O(1)$ event deletion $O(\log n)$ event insertion (links between nodes in the tree not shown). Furthermore the event tree is sorted by increasing λ, so to join clusters we just pop the first event off the tree (E_2), follow the link to the corresponding cluster (C_2), join it with the cluster below (C_3), create the new cluster (C_{new}), and update flanking events (E_1 and E_3).

An active-set descent algorithm for the ℓ_2 solutions

The proposed homotopy algorithm only gives solutions to the ℓ_1 clusterpath for identity weights, but since the ℓ_1 clusterpath in 1 dimension is a special case of the ℓ_2 clusterpath, the algorithms proposed in this section also apply to solving the ℓ_1 clusterpath with general weights.

For the ℓ_2 problem, we have the following cost function:

$$f_2(\alpha, X) = \frac{1}{2}||\alpha - X||_F^2 + \lambda\Omega_2(\alpha) \tag{2.24}$$

$$= \frac{1}{2}||\alpha - X||_F^2 + \lambda \sum_{i<j} w_{ij}||\alpha_i - \alpha_j||_2. \tag{2.25}$$

A subgradient condition sufficient for an optimal α is for all $i \in 1, ..., n$:

$$0 = \alpha_i - X_i + \lambda \sum_{\substack{j\neq i \\ \alpha_j \neq \alpha_i}} w_{ij} \frac{\alpha_i - \alpha_j}{||\alpha_i - \alpha_j||_2} + \lambda \sum_{\substack{j\neq i \\ \alpha_j = \alpha_i}} w_{ij}\beta_{ij}, \tag{2.26}$$

with $\beta_{ij} \in \mathbb{R}^p$, $||\beta_{ij}||_2 \leq 1$ and $\beta_{ij} = -\beta_{ji}$. Summing over all $i \in C$ gives the subgradient for the cluster C:

$$G_C = \alpha_C - \bar{X}_C + \frac{\lambda}{|C|} \sum_{j \notin C} w_{jC} \frac{\alpha_C - \alpha_j}{||\alpha_C - \alpha_j||_2}, \tag{2.27}$$

where $\bar{X}_C = \sum_{i\in C} X_i/|C|$ and $w_{jC} = \sum_{i\in C} w_{ij}$.

To solve the ℓ_2 clusterpath, we propose a subgradient descent algorithm, with modifications to detect cluster fusion and splitting events (Algorithm 1). Note that due to the continuity of the ℓ_2 clusterpath, it is advantageous to use warm restarts between successive calls to Solve-L2, which we do using the values of α and *clusters*.

Algorithm 1 Clusterpath-L2

Input: data $X \in \mathbb{R}^{n \times p}$, weights $w_{ij} > 0$, starting $\lambda > 0$
$\alpha \leftarrow X$
clusters $\leftarrow \{\{1\}, ..., \{n\}\}$
while $|$ *clusters* $| > 1$ **do**
  α, *clusters* $\leftarrow$ Solve-L2$(\alpha$, *clusters*$, X, w, \lambda)$
  $\lambda \leftarrow \lambda \times 1.5$
  if we are considering cluster splits **then**
    clusters $\leftarrow \{\{1\}, ..., \{n\}\}$
  end if
end while
return table of all optimal α and λ values.

Surprisingly, the ℓ_2 path is not always agglomerative, and in this case to reach the optimal solution requires restarting *clusters* $= \{\{1\}, ..., \{n\}\}$. The clusters will rejoin in the next call to Solve-L2 if necessary. This takes more time but ensures that the optimal solution is found, even if there are splits in the clusterpath, as in Figure 2.4. We compare our descent solvers' solutions to those from `cvxmod`, and note that the solver without split checks does not find the optimal solution for this pathological example.

We conjecture that there exist certain choices of w for which there are no splits in the ℓ_2 clusterpath. However, a theorem analogous to Theorem 1 that establishes necessary and sufficient conditions on w and X for splits in the ℓ_2 clusterpath is beyond the scope of this chapter. We have not observed cluster splits in our calculations of the path for identity weights $w_{ij} = 1$ and decreasing weights $w_{ij} = \exp(-\gamma||X_i - X_j||_2^2)$, and we conjecture that these weights are sufficient to ensure no splits.

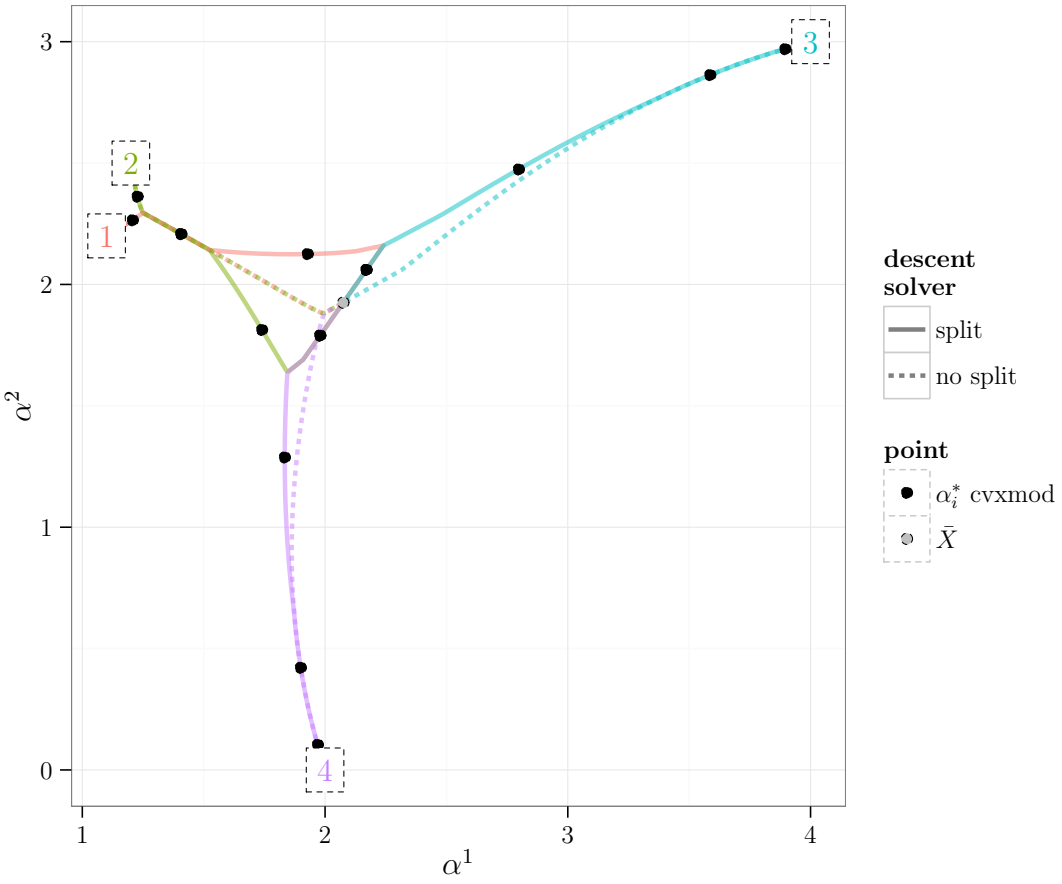

Figure 2.4: An example of a split in the ℓ_2 clusterpath for $X \in \mathbb{R}^{4\times 2}$. Data points are labeled with numbers, the CLUSTERPATH-L2 is shown as lines, and solutions α_i^* from `cvxmod` are shown as black dots. Pathological weights that cause the split were specified using $w_{12} = 9$, $w_{13} = w_{24} = 20$, and $w_{ij} = 1$ for the others.

The subgradient optimization subproblem inside of Clusterpath-L2 works as follows.

Algorithm 2 Solve-L2

Input: initial guess α, initial *clusters*, data X, weights w, regularization λ
$G \leftarrow$ Subgradient-L2$(\cdot)$
while $||G||_F^2 > \epsilon_{\text{opt}}$ **do**
  $\alpha \leftarrow$ Subgradient-Step$(\cdot)$
  α, *clusters* $\leftarrow$ Detect-Cluster-Fusion$(\cdot)$
  $G \leftarrow$ Subgradient-L2$(\cdot)$
end while
return α, *clusters*

Subgradient-L2 calculates the subgradient from (2.27), for every cluster $C \in$ *clusters*.

We developed 2 approaches to implement Subgradient-Step. In both cases we use the update $\alpha \leftarrow \alpha - rG$. With decreasing step size $r = 1/$ *iteration*, the algorithm takes many steps before converging to the optimal solution, even though we restart the iteration count after cluster fusions. The second approach we used is a line search. We evaluated the cost function at several points r and picked the r with the lowest cost. In practice, we observed fastest performance when we alternated every other step between decreasing and line search.

Detect-Cluster-Fusion calculates pairwise differences between points and checks for cluster fusions, returning the updated matrix of points α and the new list of clusters. When 2 clusters C_1 and C_2 fuse to produce a new cluster C, the coefficient of the new cluster is calculated using the weighted mean:

$$\alpha_C = \frac{|C_1|\alpha_{C_1} + |C_2|\alpha_{C_2}}{|C_1| + |C_2|}. \tag{2.28}$$

We developed 2 methods to detect cluster fusions. First, we can simply use a small threshhold on $||\alpha_{C_1} - \alpha_{C_2}||_2$, which we usually take to be some fraction of the smallest nonzero difference in the original points $||X_i - X_j||_2$. Second, to confirm that the algorithm does not fuse points too soon, for each possible fusion, we checked if the cost function decreases. This is similar to the approach used by [Friedman et al., 2007], who use a coordinate descent algorithm to optimize a cost function with an ℓ_1 fusion penalty. Although this method ensures that we reach the correct solution, it is quite slow since it requires evaluation of the cost function for every possible fusion event.

The Frank-Wolfe algorithm for ℓ_∞ solutions

We consider the following ℓ_∞ problem:

$$\min_{\alpha \in \mathbb{R}^{n \times p}} f_\infty(\alpha, X) = \frac{1}{2}||\alpha - X||_F^2 + \lambda \Omega_\infty(\alpha). \tag{2.29}$$

This problem has a piecewise linear regularization path which we can solve using a homotopy algorithm to exactly calculate all the breakpoints [Rosset and Zhu, 2007, Zhao et al., 2009]. However, empirically, the number of breakpoints in the path grows fast with p and n, leading to instability in the homotopy algorithm.

Instead, we show show that our problem is equivalent to a norm minimization over a polytope, for which an efficient algorithm exists [Frank and Wolfe, 1956].

Using the dual formulation of the ℓ_∞ norm, the regularization term is equal to:

$$\Omega_\infty(\alpha) = \sum_{i<j} w_{ij} \max_{\substack{s_{ij} \in \mathbb{R}^p \\ ||s_{ij}||_1 \leq 1}} s_{ij}^T(\alpha_i - \alpha_j).$$

Denoting by $r_i = \sum_{j>i} s_{ij} w_{ij} - \sum_{j<i} s_{ji} w_{ij} \in \mathbb{R}^p$, and by $\mathcal{R}$ the set of constraints over $R = (r_1, \ldots, r_n)$ such that the constraints over s_{ij} are respected, we have:

$$\Omega_\infty(\alpha) = \max_{R \in \mathcal{R}} \operatorname{tr}\left(R^T \alpha\right).$$

Since $\mathcal{R}$ is defined as a set of linear combinations of ℓ_1-ball inequalities, $\mathcal{R}$ is a polytope. Denoting by $Z = X - \lambda R$ and $\mathcal{Z} = \{Z \mid \frac{1}{\lambda}(X - Z) \in \mathcal{R}\}$, it is straightforward to prove that problem (2.29) is equivalent to:

$$\min_{\alpha \in \mathbb{R}^{n \times p}} \max_{Z \in \mathcal{Z}} H(\alpha, Z) = \|\alpha - Z\|_F^2 - \|Z\|_F^2,$$

where strong duality holds [Boyd and Vandenberghe, 2004]. For a given Z, the minimum of H in α is obtained by $\alpha = Z$, leading to a norm minimization over the polytope $\mathcal{Z}$.

This problem can be solved efficiently by using the Frank-Wolfe algorithm [Frank and Wolfe, 1956]. This algorithm to minimize a quadratic function over a polytope may be used as soon as it is possible to minimize linear functions in closed form. It is also known as the minimum-norm-point algorithm when applied to submodular function minimization [Fujishige et al., 2006]. In practice, it is several orders of magnitude faster than other common discrete optimization algorithms, but there is no theoretical guarantee on its complexity [Krause and Guestrin, 2009].

2.3 The spectral clusterpath

For spectral clustering, the usual formulation uses eigenvectors of the normalized Laplacian as the inputs to a standard clustering algorithm like k-means [Ng et al., 2001]. Specifically, for several values of γ, we compute a pairwise affinity matrix W such that $W_{ij} = \exp(-\gamma||X_i - X_j||_2^2)$ and a Laplacian matrix $L = D - W$ where D is the diagonal matrix such that $D_{ii} = \sum_{j=1}^{n} W_{ij}$. For each value of γ, we run k-means on the normalized eigenvectors associated with k smallest eigenvalues of L, then keep the γ with lowest reconstruction error.

Some instability in spectral clustering may come from the following 2 steps. First, the matrix of eigenvectors is formed by hard-thresholding the eigenvalues, which is unstable when several eigenvalues are close. Second, the clusters are located using the k-means algorithm, which attempts to minimize a non-convex objective. To relax these potential sources of instability, we propose the "spectral clusterpath," which replaces (a) hard-thresholding by soft-thresholding and (b) k-means by the clusterpath.

Concretely, we call $(\Lambda_i)_{1\leq i\leq n}$ the nontrivial eigenvalues sorted in ascending order, and we write the matrix of transformed eigenvectors to cluster as VE, where V is the full matrix of sorted nontrivial eigenvectors and E is the diagonal matrix such that $E_{ii} = e(\Lambda_i)$, and $e : \mathbb{R} \to \mathbb{R}$ ranks importance of eigenvectors based on their eigenvalues. Standard spectral clustering takes $e_{01}(x) = 1_{x\leq\Lambda_k}$ such that only the first k eigenvalues are selected. This is a non-convex hard-thresholding of the full matrix of eigenvectors. We propose the exponential function $e_{\exp}(x) = \exp(-\nu x)$, with $\nu > 0$, as a convex relaxation.

2.4 Results

Our model poses 3 free parameters to choose for each matrix to cluster: norm, weights, and regularization. On one hand, this offers the flexibility to tailor the geometry of the solution path and number of clusters for each data set. On the other hand, this poses model selection problems as training clustering models is not straightforward. Many heuristics have been proposed for automatically choosing the number of clusters [Tibshirani et al., 2001], but it is not clear which of these is applicable to any given data set.

In the experiments that follow, we chose the model based on the desired geometry of the solution path and number of clusters. We generally expect rotation invariance in multivariate clustering models, so we chose the ℓ_2 norm with Gaussian weights to encourage sensitivity to local density.

Verification on non-convex clusters

To compare our algorithm to other popular methods in the setting of non-convex clusters, we generated data in the form of 2 interlocking half-moons (Figure 2.5), which we used as input for several clustering algorithms (Table 2.1). We used the original data as input for k-means, Gaussian mixtures, average linkage hierarchical clustering, and the ℓ_2 clusterpath with $\gamma = 2$. For the other methods, we use the eigenvectors from spectral clustering as input. Each algorithm uses 2 clusters and performance is measured using the normalized Rand index, which varies from 1 for a perfect match to 0 for completely random assignment [Hubert and Arabie, 1985].

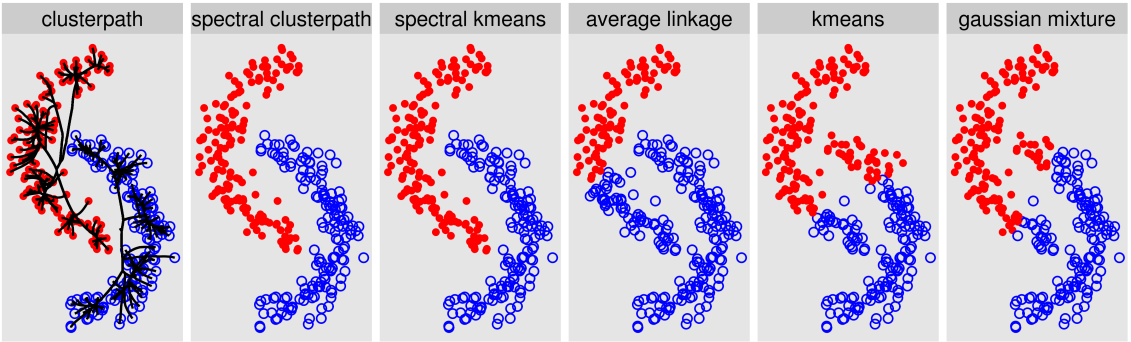

Figure 2.5: Typical results for 6 clustering algorithms applied to 2 half-moon non-convex clusters. The ℓ_2 clusterpath tree learned from the data is also shown. Spectral clustering and the clusterpath correctly identify the clusters, while average linkage hierarchical clustering and k-means fail.

In the original input space, hierarchical clustering and k-means fail, but the clusterpath is able to identify the clusters as well as the spectral methods, and has the added benefit of learning a tree from the data. However, the clusterpath takes 3-10 times more time than the spectral methods. Of the methods that cluster the eigenvectors, the most accurate 2 methods use e_{exp} rather than e_{01}, providing evidence that the convex relaxation stabilizes the clustering.

method	mean.rand	sd.rand	mean.seconds	sd.seconds
e_{exp} spectral clusterpath	1.00	0.01	8.28	3.09
e_{exp} spectral kmeans	1.00	0.01	2.57	0.19
e_{01} spectral kmeans	0.91	0.20	2.57	0.18
e_{01} Ng et al. kmeans	0.95	0.19	5.52	0.44
clusterpath	0.96	0.13	28.55	2.24
kmeans	0.27	0.05	0.00	0.00
average linkage	0.40	0.13	0.01	0.00
gaussian mixture	0.43	0.13	0.08	0.04

Table 2.1: Mean and standard deviation of performance and timing of several clustering methods on identifying 20 simulations of the half-moons in Figure 2.5. Ng et al. uses $\tilde{L} = I - D^{-1/2}WD^{-1/2}$ rather than $L = D - W$ as discussed in the text.

Recovery of many Gaussian clusters

We also tested our algorithm in the context of 25 Gaussian clusters arranged in a 5×5 grid in 2 dimensions. As shown in Figure 2.6, 20 data points were generated from each cluster, and the resulting data were clustered using k-means, hierarchical clustering, and the weighted ℓ_2 clusterpath.

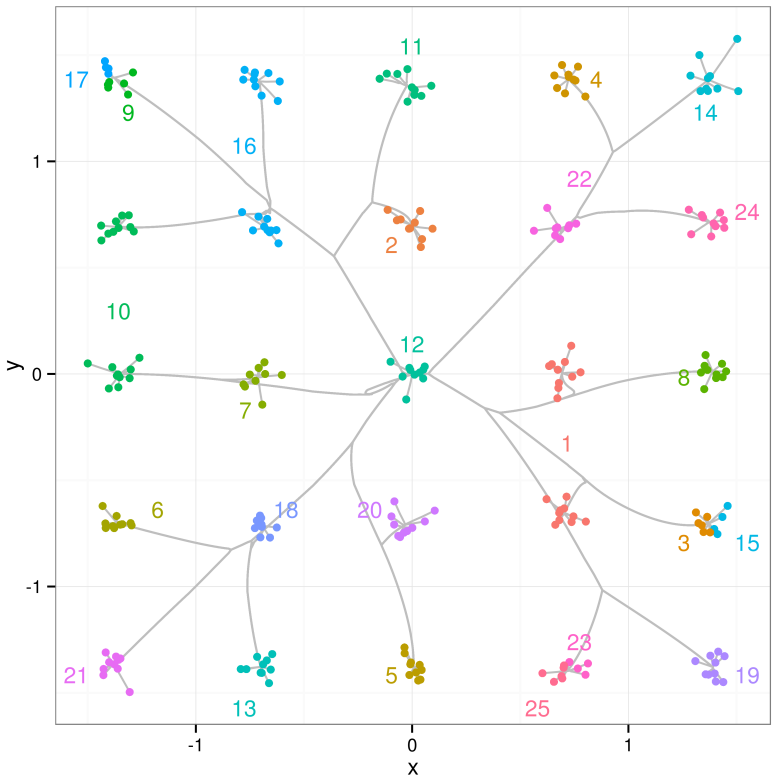

Figure 2.6: A grid of gaussian clusters breaks k-means but not the clusterpath. The result from k-means clustering with $k = 25$ is shown in color with numbered labels, and the ℓ_2 clusterpath tree with $\gamma = 10$ is shown in grey.

As shown in Table 2.2, the clusterpath performs similarly to hierarchical clustering, which exactly recovers the clusters, and k-means fails. Thus, the clusterpath may be useful for clustering tasks that involve many clusters.

Clustering method	Rand	SD
kmeans	0.8365	0.0477
clusterpath	0.9955	0.0135
average linkage hierarchical	1.0000	0.0000

Table 2.2: Performance of several clustering methods on identifying a grid of Gaussian clusters. Means and standard deviations from 20 simulations are shown.

Application to clustering the iris data

To evaluate the clusterpath on a nontrivial task, we applied it to the scaled iris data. In Figure 2.7, we show a scatter plot matrix of the 4-dimensional iris data with the ℓ_2 clusterpath using weight parameter $\gamma = 1$.

We compared clusterpath to other common clustering methods using the iris and the half-moons data sets, and report the results in Figure 2.8. We calculated a series of clusterings using each method and measured performance of each using the normalized Rand index.

The iris data have 3 classes, of which 2 overlap, so the Gaussian Mixture Model is the only algorithm capable of accurately detecting these clusters when $k = 3$. These data suggest that the clusterpath is not suitable for detecting clusters with large overlap. However, performance is as good as hierarchical clustering, less variable than k-means, and more stable as the number of clusters increases.

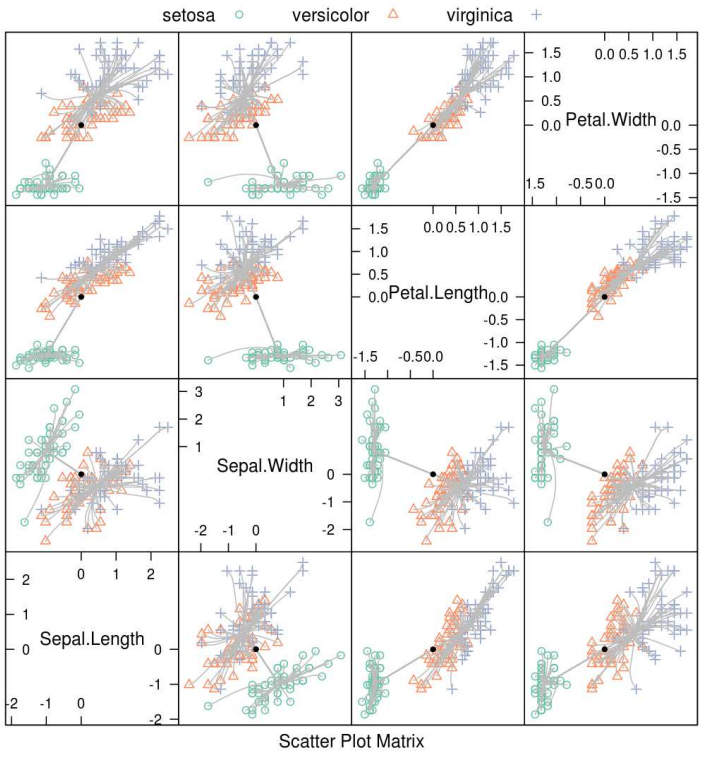

Figure 2.7: The 3-class, 4-dimensional iris data are plotted using colored symbols, with the mean shown as a black dot. The ℓ_2 clusterpath using weight parameter $\gamma = 1$ is shown in grey.

Also note that the clusterpath accuracy on the moons data increases as we increase the weight parameter γ.

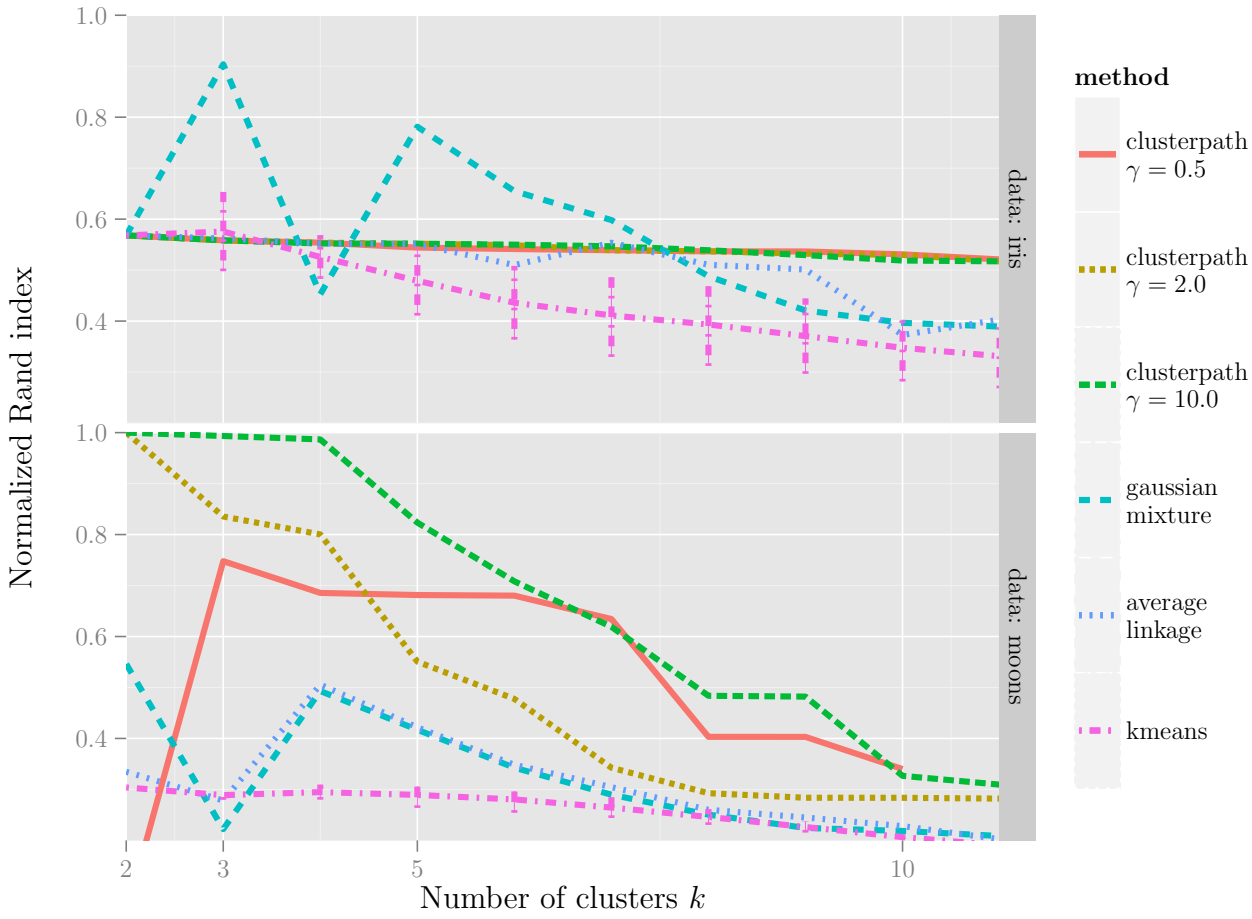

Figure 2.8: Performance on the iris and moons data, as measured by the normalized Rand index of models with 2-11 clusters. The weighted ℓ_2 clusterpath was calculated using 3 different Gaussian weight parameters γ, and we compare with Gaussian mixture models, hierarchical clustering with average linkage, and k-means.

2.5 Conclusions

We proposed a family of linear models using several convex pairwise fusion penalties which result in hierarchical regularization paths useful for clustering. The ℓ_1 path-following homotopy algorithm easily scales to thousands of points. The other proposed algorithms can be directly applied to hundreds of points, and could be applied to larger data sets by, for example, adding a preprocessing step using k-means. The ℓ_1 and ℓ_2 solvers were implemented in R and C++, and are available in the clusterpath package on R-Forge:

`https://r-forge.r-project.org/projects/clusterpath/`

The ℓ_∞ solver was written in MATLAB and is available on the web page of Armand Joulin:

`http://www.di.ens.fr/~joulin/code/clusterpath_norm_Inf.zip`

Our experiments demonstrated the flexibility of the ℓ_2 clusterpath for the unsupervised learning of non-convex clusters, large numbers of clusters, and hierarchical structures. We also observed that relaxing hard-thresholding in spectral clustering is useful for increasing clustering accuracy and stability. For the iris data, the clusterpath performed as well as hierarchical clustering, and is more stable than k-means.

We proved that the identity weights are sufficient for the ℓ_1 clusterpath to be strictly agglomerative. Establishing necessary and sufficient conditions on the weights for the ℓ_2 problem is an avenue for further research.

To extend these results, we are currently pursuing research into optimizing a linear model with a non-identity design matrix and the clusterpath penalty. We note that there could be a future application for the algorithms presented in this chapter in solving the proximal operator, which is the same as (2.4) for the clusterpath penalty.

Chapter 3

Segmentation model selection using visual annotations

In this chapter I will discuss how visual annotations may be used for segmentation model selection. This work has been described in a technical report [Hocking et al., 2012], and is submitted to a journal for peer review.

Chapter summary

Many models have been proposed to detect copy number alterations in chromosomal copy number profiles, but it is usually not obvious to decide which is most effective for a given data set. Furthermore, most methods have a smoothing parameter that determines the number of breakpoints and must be chosen using various heuristics.

We present three contributions for copy number profile smoothing model selection. First, we propose to select the model and degree of smoothness that maximizes agreement with visual breakpoint region annotations. Second, we develop cross-validation procedures to estimate the error of the trained models. Third, we apply these methods to compare many existing models on a new database of annotated neuroblastoma copy number profiles, which we make available in R package **neuroblastoma** as a public benchmark for testing new algorithms. Whereas previous studies have been qualitative or limited to simulated data, our approach is quantitative and suggests which algorithms are most accurate in practice on real data.

The annotated copy number profiles are in R package neuroblastoma:
`http://cran.r-project.org/web/packages/neuroblastoma/index.html`
To annotate other data sets, we developed GUI tools:
Simple Python script: `http://pypi.python.org/pypi/annotate_regions`
Interactive modeling web site: `https://bioviz.rocq.inria.fr/plotter/`

3.1 Introduction and related work

DNA copy number alterations (CNAs) can result from various types of genomic rearrangements, and are important in the study of many types of cancer [Weinberg, 2006]. In particular, clinical outcome of patients with neuroblastoma has been shown to be worse for tumors with segmental alterations or breakpoints in specific genomic regions [Janoueix-Lerosey et al., 2009, Schleiermacher et al., 2010]. Thus, to construct an accurate predictive model of clinical outcome for these tumors, we must first accurately detect the precise location of each breakpoint.

In recent years, array comparative genomic hybridization (aCGH) microarrays have been developed as genome-wide assays for CNAs, using the fact that microarray fluoresence intensity is proportional to DNA copy number [Pinkel et al., 1998]. In parallel, there have been many new mathematical models proposed to smooth the noisy signals from these microarray assays in order to recover the CNAs [Willenbrock and Fridlyand, 2005]. Each model has different assumptions about the data, and it is not obvious to decide which model is appropriate for a given data set.

Furthermore, most models have parameters that control the degree of smoothness. Varying these smoothing parameters will vary the number of detected breakpoints. Most authors give default values that accurately detect breakpoints on some data, but do not necessarily generalize well to other data. There are some specific criteria for choosing the degree of smoothness in some models [Lavielle, 2005, Zhang and Siegmund, 2007, Zhang et al., 2010], but the mathematical assumptions of these models are not often verified in real noisy microarray data, which can lead to poor estimation of CNAs.

In practice, visualization tools such as VAMP [La Rosa et al., 2006] are often used to plot the normalized microarray signals against genomic position for interpretation by an expert biologist looking for CNAs. Indeed, to motivate the use of their cghFLasso smoothing model, Tibshirani and Wang write "The results of a CGH experiment are often interpreted by a biologist, but this is time consuming and not necessarily very accurate" [Tibshirani and Wang, 2007].

In contrast, this chapter takes the opposite view and assumes that the expert interpretation of the biologist is very accurate. The first contribution of this chapter is a smoothing model training protocol based on this assumption. To tune the smoothness parameter in practice, the biologist will often examine plots of the microarray signal with a smoothed model, changing the smoothness parameter until the model seems to capture all the visible breakpoints the data. In the Methods section, we make this

intuition concrete by defining a training protocol based on visual annotations that can quantify the accuracy of a smoothing method. We note that using databases of visual annotations is not a new idea, and has been used successfully for object recognition in photos and cell phenotype recognition in microscopy [Russell et al., 2008, Jones et al., 2009]. In array CGH analysis, Shah et al. [2006] attempted to model prior knowledge of locations of CNAs, but no previous models have been designed to exploit the annotated regions that we introduce in this chapter.

Our second contribution is a protocol to estimate the breakpoint detection ability of the trained smoothing models on real data. In the Methods section, we propose to estimate the false positive and false negative rates of the trained models using cross-validation. This provides a quantitative criterion for deciding which smoothing algorithms are appropriate breakpoint detectors for which data.

The third contribution of this chapter is a systematic, quantitative comparison of the accuracy of several common smoothing algorithms on a new database of 575 annotated neuroblastoma copy number profiles, which we give in the Results section. There are several publications which attempt to assess the accuracy of smoothing algorithms, and these methods fall into 2 categories: simulations and low-throughput experiments. GLAD, DNAcopy, and a hidden Markov model were compared by examining false positive and false negative rates for detection of a breakpoint at a known location in simulated data [Willenbrock and Fridlyand, 2005]. However, there is no way to verify if the assumptions of the simulation hold in a real data set, so the value of the comparison is limited. In another article, the accuracy of the CNVfinder algorithm was assessed using quantitative PCR [Fiegler et al., 2006]. But quantitative PCR is low-throughput and costly, so is not routinely done as a quality control. So in fact there are no previous studies that quantitatively compare breakpoint detection of smoothing models on real data. In this chapter we propose to use annotated regions instead of precise breakpoint locations for quantifying smoothing model accuracy, and we make available 575 new annotated neuroblastoma copy number profiles as a benchmark for the community to test new algorithms on real data.

Several authors have recently proposed methods for so-called joint segmentation of multiple CGH profiles, under the hypothesis that each profile shares breakpoints in the exact same location [Vert and Bleakley, 2010, Ritz et al., 2011]. These models are not useful in our setting, since we assume that breakpoints do not occur in the exact same locations across copy number profiles. Instead, we learn a model that will accurately detect a different number of breakpoints in each copy number profile.

3.2 Methods

Assume that we have observed n chromosomal copy number profiles, each with an unknown number of breakpoints at unknown locations. We would like to recover the breakpoints accurately using a model with parameter λ that controls the degree of smoothness. As shown in Figure 3.1, we represent the d probes on chromosome c of profile i using the following numbers:

$$\begin{array}{ccccccc} p_1 & \leq & \ldots & \leq & p_d & \in \mathbb{N} & \text{positions on chromosome } c \\ y_1 & , & \ldots & , & y_d & \in \mathbb{R} & \text{logratio measurements} \\ \hat{y}_1^\lambda & , & \ldots & , & \hat{y}_d^\lambda & \in \mathbb{R} & \text{smoothed profile} \end{array}$$

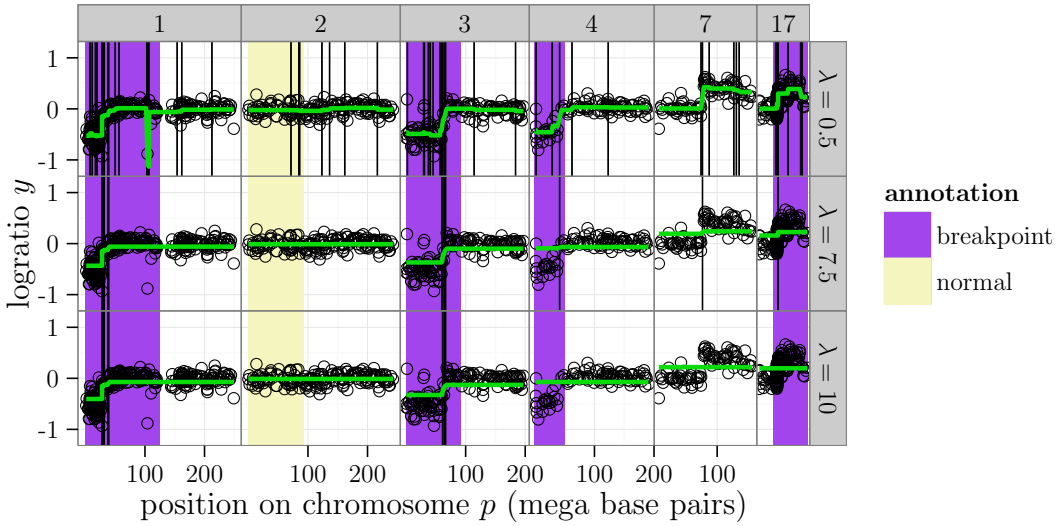

Figure 3.1: Model agreement to annotated regions can be measured by examining the positions of predicted breakpoints $\hat{B}_i^\lambda$ (vertical black lines) observed in the smoothing model $\hat{y}^\lambda$ (green lines). Black circles show logratio measurements y plotted against position p for a single profile $i = 375$. Chromosomes are shown in panels from left to right, and different values of the smoothing parameter λ in the flsa model are shown in panels from top to bottom. Models with too many breakpoints ($\lambda = 0.5$) and too few breakpoints ($\lambda = 10$) are suboptimal, so we pick an intermediate model ($\lambda = 7.5$) that maximizes agreement with the annotations, thus detecting a new breakpoint on chromosome 7 which was not annotated.

We define the breakpoints predicted on this chromosome as the set of positions after which there is a jump in the smoothed signal:

$$\hat{B}_{ic}^{\lambda} = \left\{ \lfloor (p_j + p_{j+1})/2 \rfloor \mid \hat{y}_j^{\lambda} \neq \hat{y}_{j+1}^{\lambda},\ \forall j = 1, \ldots, d-1 \right\} \tag{3.1}$$

Then, we define $\hat{B}_i^{\lambda}$ to be the complete set of genomic breakpoints predicted by parameter λ for profile i, over all chromosomes c.

Breakpoint annotations quantify model accuracy

Intuitively, by visual inspection of the noisy signal, it is not obvious to locate the exact location of a breakpoint, but it should be easy to determine whether or not a region contains a breakpoint. So rather than defining annotations in terms of precise breakpoint locations, we instead define them in terms of regions. We define a genomic region $R_k = [\underline{r}_k, \overline{r}_k]$ as the interval of base pairs between the min $\underline{r}_k$ and max $\overline{r}_k$.

So, we define the breakpoint annotation for profile i in region k as

$$b_{ik} = \begin{cases} 0 & \text{if profile } i \text{ has no breakpoints in } R_k \\ 1 & \text{if profile } i \text{ has at least 1 breakpoint in } R_k, \end{cases} \tag{3.2}$$

which can be determined by visual inspection of the scatterplot of logratio measurements y versus position p, as in Figure 3.1.

The idea for model selection is to choose λ such that the predicted breakpoints $\hat{B}_i^{\lambda}$ agree with the annotations b_{ik}, as shown in Figure 3.1. To quantify this, for each region k, we predict 0 if there are no predicted breakpoints in the region, and 1 if there is at least 1 predicted breakpoint:

$$\hat{b}_{ik}^{\lambda} = \begin{cases} 0 & \text{if } R_k \cap \hat{B}_i^{\lambda} = \emptyset \\ 1 & \text{otherwise} \end{cases} \tag{3.3}$$

We can measure the error of a model at region k on profile i with the indicator function

$$E_i^k(\lambda) = \begin{cases} 0 & \text{if } b_{ik} = \hat{b}_{ik}^{\lambda} \\ 1 & \text{otherwise.} \end{cases} \tag{3.4}$$

and with respect to an entire profile i using

$$E_i^{\text{local}}(\lambda) = \sum_k E_i^k(\lambda). \tag{3.5}$$

We define the local model as the model obtained by choosing a different $\hat{\lambda}_i$ to minimize Equation 3.5 for each profile i, as shown in the middle and bottom rows of Figure 3.2.

We can also learn a globally optimal smoothing parameter λ by minimizing the error with respect to all the profiles:

$$E^{\text{global}}(\lambda) = \sum_{i=1}^{n} E_i^{\text{local}}(\lambda). \tag{3.6}$$

The global model $\hat{\lambda}$ minimizes Equation 3.6, as shown in the top row of Figure 3.2.

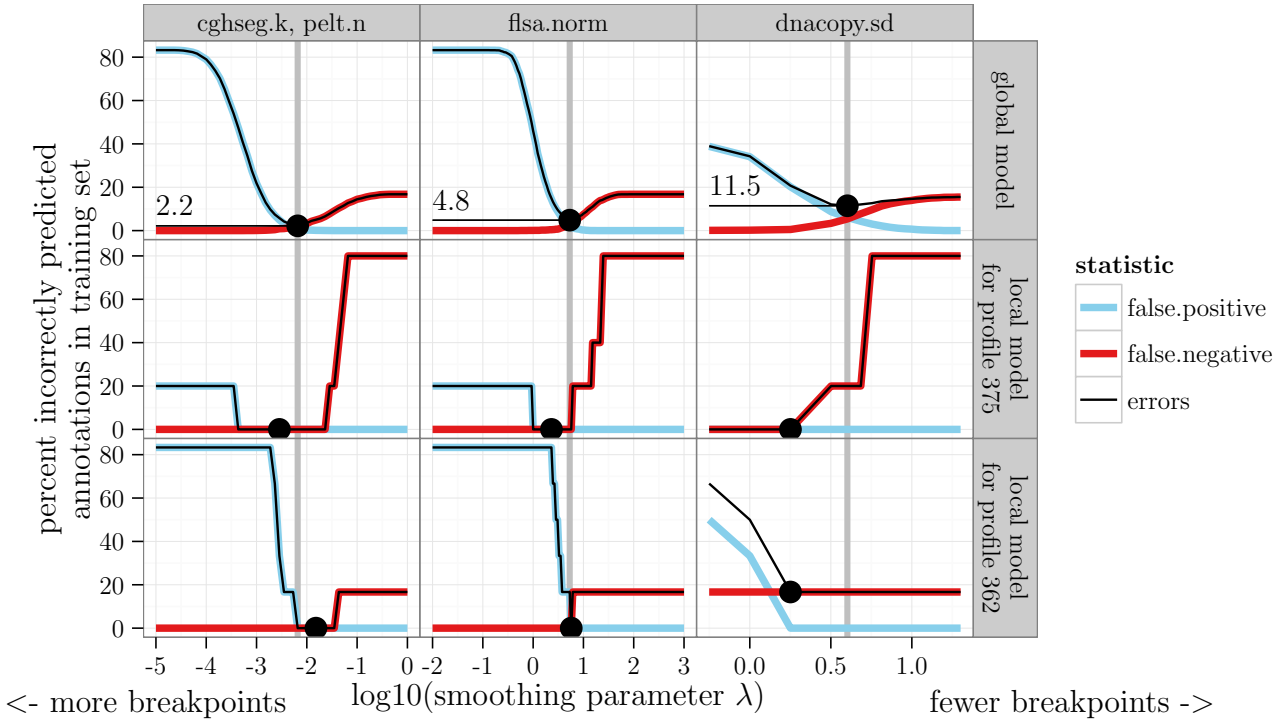

Figure 3.2: Training error functions plotted against smoothing parameter λ. In the top row, we plot $E^{\text{global}}(\lambda)$ from Equation 3.6, and in the other rows, we plot $E_i^{\text{local}}(\lambda)$ from Equation 3.5. Each column of plots shows the error of a particular algorithm, and the minimum chosen using the global training procedure is shown using a vertical grey line. Note that the local model training error can be reduced by moving from the globally optimal smoothing parameter $\hat{\lambda}$ to a local value $\hat{\lambda}_i$, as in profile $i = 375$ for dnacopy.sd. For the local models trained on single profiles, there are at most 6 training examples. So many smoothing parameters attain the minimum, and the model we select is shown as a black dot.

Picking the optimal degree of smoothness

We assume that λ is a tuning parameter that is monotonic in the number of breakpoints, which is the case for the models considered in this chapter.

Fix a set of smoothing parameters, and run the smoothing algorithm with each of these parameters. Intuitively, we should pick the value of λ that maximizes agreement with annotation data. For global models, we attempt to minimize Equation 3.6, and there is usually one best value, λ^*.

However, for the local model for profile i, we want to minimize the local error as defined in Equation 3.5. Since the training set consists of only the annotations of one profile i, there may be no unique smoothing parameter λ that minimizes the error. We propose to pick between models that achieve the minimum number of errors based on the shape of the error curve, and these cases are illustrated in Figure 3.2.

1. When the minimum error is achieved in a range of intermediate parameter values, we pick a value in the middle. This occurs in the local error curves shown for flsa.norm and cghseg.k.

2. When the minimum is attained by the model with the most breakpoints, we pick the model with the fewest breakpoints that has the same error. This attempts to minimize the false positive rate, and occurs for profile $i = 375$ with model dnacopy.sd.

3. When the minimum is attained by the model with the fewest breakpoints, we pick the model with the most breakpoints that has the same error. This attempts to minimize the false negative rate, and occurs for profile $i = 362$ with model dnacopy.sd.

Leave-one-out cross-validation for comparing local and global models

To compare the breakpoint detection performance of local and global models on un-annotated regions, we propose leave-one-out cross-validation on regions.

- For each annotated region k:

    1. Designate R_k as the test region, and set aside the annotations in this region from all the profiles.
    2. Using all the other annotations as a training set, pick the best λ using the protocol described in Section 3.2. For local models

we learn a profile-specific λ_i that minimizes E_i^{local}, and for global models we learn a global λ that minimizes E^{global}.

3. To estimate how the model generalizes, count the errors of the learned model in the test region R_k.

- To estimate the ability of the trained model to predict breakpoints at a general un-annotated region, take the mean test error over all regions.

n/t-fold cross-validation to estimate error on un-annotated profiles

Since the annotation process is time-consuming, we are interested in training an accurate breakpoint detector with as few annotations as possible. Thus we would like to answer the following question: how many profiles t do I need to annotate before I get a global model that will generalize well to all the other profiles?

To answer this question, we estimate the error of a global model trained on the annotations from t profiles using cross-validation. We divide the set of n annotated profiles into exactly $\lfloor n/t \rfloor$ folds, each with approximately t profiles. For each fold, we consider its annotations a training set for a global model, and combine the other folds as a test set to quantify the model error. The final estimate of generalization error is then the average model error over all folds.

Normal	Breakpoint annotations						
annotations	0	1	2	3	4	5	6
0	0	0	0	1	0	0	2
1	0	0	0	0	3	9	0
2	0	0	0	5	29	0	0
3	0	1	3	60	0	0	0
4	0	8	64	0	0	0	0
5	8	47	0	0	0	0	0
6	335	0	0	0	0	0	0

Table 3.1: Counts of profiles in the neuroblastoma data set, conditional on number of annotations. Note that most profiles have more normal regions than breakpoint regions. For example, 335 profiles have all 6 regions annotated as normal.

Data: neuroblastoma copy number profiles

We analyzed a new data set of $n = 575$ copy number profiles from aCGH microarray experiments on neuroblastoma tumors taken from patients at diagnosis. The microarrays were produced using various technologies, so do not all have the same probes. The number of probes per microarray varies from 1719 to 71340. In this chapter we analyzed the normalized logratio measurements of these microarrays, which we have made available as `neuroblastoma$profiles` in R package **neuroblastoma** on CRAN:

`http://cran.r-project.org/web/packages/neuroblastoma/index.html`

Six chromosome arms known to be associated with prognostic impact were annotated in the microarray data set [Janoueix-Lerosey et al., 2009]. Each region was defined by the start and end of a chromosome arm, and the genomic coordinates of these regions are given in Table 3.2.

For each profile i, our domain expert annotated each region k by examining the plotted profile in VAMP [La Rosa et al., 2006] and recording 0 or 1 in a spreadsheet, according to the definition of breakpoint annotations in Equation 3.2. Table 3.2 shows counts of annotations per region, and Table 3.1 shows counts of annotations per profile. Some profiles have less than 6 annotations since we excluded regions where presence of breakpoints could not be determined by visual inspection. The annotations are shown as colored rectangles in Figure 3.1, and are available as `neuroblastoma$annotations` in R package **neuroblastoma** on CRAN.

min $= \underline{r}_k$	max $= \overline{r}_k$	chrom c_k	breakpoint	normal	(all)
0.0	125.0	1	103	464	567
0.0	93.3	2	110	464	574
0.0	91.0	3	43	531	574
0.0	50.4	4	35	534	569
53.7	135.0	11	107	464	571
24.0	81.2	17	175	388	563
		(all)	573	2845	3418

Table 3.2: Counts of normal and breakpoint annotations in the neuroblastoma data set, conditional on region. Min and max limits of each region are shown in mega base pairs.

Algorithms: copy number profile smoothing models

In this study we considered smoothing models from the bioinformatics literature with free software implementations available as R packages on CRAN [R Development Core Team, 2012], R-Forge [Theußl and Zeileis, 2009], or Bioconductor [Gentleman et al., 2004]. We systematically implemented and analyzed each of these models using the code published in R package **bams**, for Breakpoint Annotation Model Smoothing:

`http://cran.r-project.org/web/packages/bams/index.html`

We used version 1.0 of the **gada** package from R-Forge to calculate the sparse Bayesian learning model of Pique-Regi et al. [2008]. We varied the degree of smoothness by adjusting the `T` parameter of the `BackwardElimination` function, and for the gada.default model, we did not use the `BackwardElimination` function.

We used version 1.03 of the **flsa** package from CRAN to calculate the Fused Lasso Signal Approximator as described by Hoefling [2009]. The FLSA solves the following optimization problem for each chromosome:

$$\hat{y}^\lambda = \arg\min_{\mu \in \mathbb{R}^d} \frac{1}{2} \sum_{i=1}^{d} (y_i - \mu_i)^2 + \lambda_1 \sum_{i=1}^{d} |\mu_i| + \lambda_2 \sum_{i=1}^{d-1} |\mu_i - \mu_{i+1}|. \qquad (3.7)$$

We define a grid of values $\lambda \in \{10^{-5}, \ldots, 10^{12}\}$, take $\lambda_1 = 0$, and consider the following parameterizations for λ_2:

- flsa: $\lambda_2 = \lambda$.
- flsa.norm: $\lambda_2 = \lambda d \times 10^6 / l$ where d is the number of points and l is the length of the chromosome in base pairs.

We used version 1.29.0 of the **DNAcopy** package from Bioconductor to fit the circular binary segmentation model of Venkatraman and Olshen [2007]. We varied the degree of smoothness by adjusting the `undo.SD`, `undo.prune`, and `alpha` parameters of the `segment` function. However, the dnacopy.prune algorithm was too slow ($>$ 24 hours) for some of the profiles with many data points, so these profiles were excluded from the analysis of dnacopy.prune.

We used version 0.2-1 of the **cghFLasso** package from CRAN, which implements the method of Tibshirani and Wang [2007], but does not provide any smoothness parameters for breakpoint detection.

We used version 2.17.0 of the **GLAD** package from Bioconductor to fit the GLAD adaptive weights smoothing model of Hupé et al. [2004]. We varied the degree of smoothness by adjusting the `lambdabreak` and `MinBkpWeight` parameters of the `daglad` function. For the glad.haarseg model,

we used the `smoothfunc="haarseg"` option and varied the `breaksFdrQ` parameter to fit the wavelet smoothing model of Ben-Yaacov and Eldar [2008].

We used version 0.01 of the **cghseg** package from R-Forge to fit the maximum-likelihood piecewise constant smoothing model of Picard et al. [2005] for each chromosome using pruned dynamic programming [Rigaill, 2010]. For the cghseg.mBIC model, we used the modified Bayesian information criterion described by Zhang and Siegmund [2007], which has no smoothness parameter, and is implemented in the `uniseg` function of the **cghseg** package. For the cghseg.k model, we used the `segmeanCO` function with `kmax=20` to obtain the maximum-likelihood piecewise constant smoothing model $\mu^k \in \mathbb{R}^d$ for $k = 1, \ldots, 20$ segments. Lavielle [2005] suggested penalizing k breakpoints in a signal sampled at d points using λk, and varying λ as a tuning parameter. We implemented this model selection criterion as the cghseg.k model, for which we define the optimal number of segments

$$k^*(\lambda) = \underset{k \in \{1,\ldots,20\}}{\arg\min}\ \lambda k + \frac{1}{d}\sum_{i=1}^{d}(y_i - \mu_i^k)^2, \tag{3.8}$$

and the optimal smoothing $\hat{y}^\lambda = \mu^{k^*(\lambda)}$.

We used the `cpt.mean` function in vesion 0.7 of the **changepoint** package from CRAN to fit a penalized maximum likelihood model using a Pruned Exact Linear Time (PELT) algorithm [Killick et al., 2011]. PELT defines μ^k in the same way as cghseg, but defines the optimal number of segments as

$$k^*(\beta) = \underset{k \in \{1,\ldots,d\}}{\arg\min}\ \beta(k-1) + \sum_{i=1}^{d}(y_i - \mu_i^k)^2. \tag{3.9}$$

For the pelt.default model, we used the default settings which specify `penalty="SIC"` for the Schwarz or Bayesian Information Criterion, meaning $\beta = \log d$. For the pelt.n model, we specified `penalty="Manual"` which means that the `value` parameter is used as β, and the `cpt.mean` function returns $\mu^{k^*(\beta)}$. We defined the same grid of λ values that we used for cghseg.k, and let $\beta = \lambda d$. Note that this model is mathematically equivalent to cghseg.k.

3.3 Results and discussion

All the smoothing models described in the Algorithms section were applied to all the annotated neuroblastoma copy number profiles, described in the Data section. Note that to decrease computation time, the model fitting may be trivially parallelized for profiles, algorithms, and smoothing parameter values.

After fitting the models, we used the breakpoint annotations to quantify the accuracy of each model. We calculated the local and global error curves $E(\lambda)$, which quantify how many breakpoint annotations disagree with the model breakpoints, as a function of the model smoothness parameter λ. The global model is defined as the smoothness parameter $\hat{\lambda}$ that maximizes agreement with the breakpoint annotations from all profiles. In contrast, for every profile i, we define the local model as the smoothness parameter $\hat{\lambda}_i$ that maximizes agreement with the breakpoint annotations from profile i.

Among global models, cghseg.k and pelt.n exhibit the smallest training error in the neuroblastoma data

Training error curves for cghseg.k, flsa.norm, dnacopy.sd, and glad.lambdabreak are shown in Figure 3.2. Note that the global curves do not achieve zero training error but the local curves often do, suggesting that the local training strategy may be useful for decreasing the training error. Also note the inflexibility of the dnacopy.sd model, which does not detect a breakpoint in profile $i = 362$, even at the smallest parameter value, corresponding to the model with the most breakpoints. Finally, note the minimum error of 2.2% achieved by cghseg.k and pelt.n, the global models with the smallest training error.

The ROC curves for the training error of the global models for each algorithm are traced in Figure 3.3. It is clear that the default parameters of each algorithm except PELT show relatively large false positive rates. In contrast, the pelt.default model and the models chosen by maximizing agreement with the breakpoint annotations exhibit smaller false positive rates at the cost of smaller true positive rates. The ROC curves suggest that the cghseg.k and pelt.n models are the most discriminative for breakpoint detection in the neuroblastoma data.

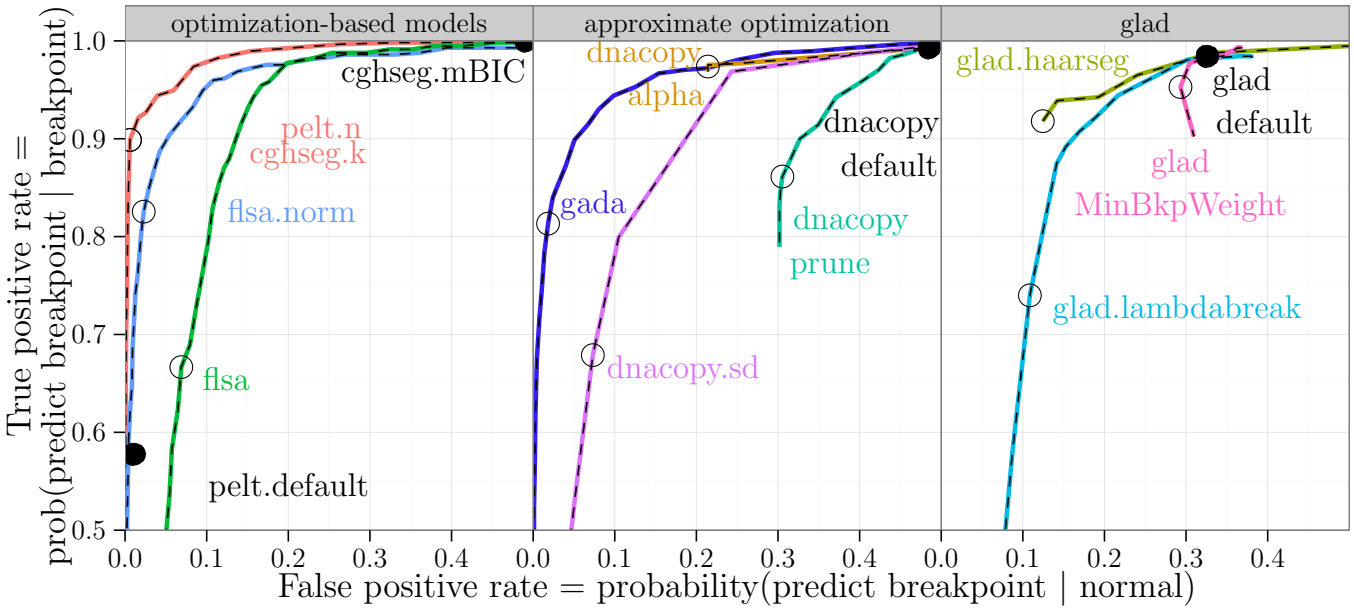

Figure 3.3: ROC curves for the training error with respect to the breakpoint annotation data are shown as colored lines. The curves are shown in 3 panels zoomed to the upper left region of ROC space to avoid visual clutter. Each curve is traced by plotting the error of a model as the degree of smoothness is varied, and an empty black circle shows the global model chosen by minimizing the error with respect to all annotations. Algorithms with no tuning parameters are shown as black dots. Note that some ROC curves appear incomplete since some segmentation algorithms are not flexible enough for the task of breakpoint detection, even though we ran each algorithm on a very large range of smoothness parameter values.

Global models generalize better than local models

The leave-one-out cross-validation protocol was used to contrast the test error of the global and local models. Table 3.3 shows the error, false positive, and false negative rates of each model, averaged over the 6 test regions.

It is clear that the training procedure makes no difference for models glad.default, dnacopy.default, cghseg.mBIC, pelt.default, gada.default, and cghFLasso, which have no smoothness parameters. The large error of these models suggest that the assumptions of their default parameter values do not hold in the neuroblastoma data set. More generally, these error rates suggest that smoothness parameter tuning is critically important to obtain an accurate smoothing of real copy number profiles.

For dnacopy.prune, glad.MinBkpWeight, glad.lambdabreak, and flsa, there appears to be little difference between the local and global training procedures. For models flsa.norm, gada, pelt.n, and cghseg.k, there seems to be a clear advantage for the global models which share information between profiles. The equivalent cghseg.k and pelt.n models show the minimal estimated test error of only 2.2% on these data. The slight differences between the local FP and FN rates of cghseg.k and pelt.n can be attributed to rounding errors when specifying `cpt.mean(value=sprintf("n*%f",lambda))` for pelt.n.

Finally, note that the false positive rate of locally trained models is higher than the false negative rate for most algorithms. This can be explained by the larger fraction of normal annotations present in the training set, and the fact that many profiles have only normal annotations (Table 3.1).

Timing PELT and cghseg

Interestingly, the cghseg.k method is somewhat faster than pelt.n, since these models use different algorithms to calculate the same segmentation. For cghseg.k, we use pruned dynamic programming to calculate the best segmentation μ^k for $k \in \{1, \ldots, 20\}$ segments, which is the slow step. Then, we calculate the best segmentation for $\lambda \in \{\lambda_1, \ldots, \lambda_{100}\}$, based on the stored μ^k values. In contrast, the Pruned Exact Linear Time algorithm must be run for each $\lambda \in \{\lambda_1, \ldots, \lambda_{100}\}$, and there is no information shared between λ values.

However, timing the PELT and cghseg methods without tuning parameters shows the opposite trend. In particular, the default cghseg.mBIC method is slower than the pelt.default method. This makes sense since cghseg must first calculate the best segmentation μ^k for several k, then use the

mBIC criterion to choose among them. In contrast, the PELT algorithm recovers just the μ^k which corresponds to the Schwarz Information Criterion penalty constant $\beta = \log d$. So if you want to use a particular penalty constant β instead of the annotation-guided approach we suggest in this chapter, the default PELT method offers a modest speedup over cghseg.

	Global			Local			Timings
	errors	FP	FN	errors	FP	FN	seconds
cghseg.k ■	2.2	0.6	11.6	11.1	13.0	7.0	1.69
pelt.n ■	2.2	0.6	11.6	11.1	12.9	6.9	5.83
gada ■	6.5	4.1	19.8	12.9	14.4	10.4	6.36
flsa.norm ■	6.7	3.6	18.5	14.8	15.2	10.5	0.08
pelt.default	8.0	1.1	44.8	8.0	1.1	44.8	0.19
dnacopy.sd ■	11.5	7.6	32.2	12.8	10.0	28.8	51.62
glad.haarseg ■	11.8	12.6	8.0	17.8	20.7	4.1	29.51
glad.lambdabreak ■	14.1	12.3	23.0	15.8	16.2	14.2	14.44
flsa ■	16.0	12.7	36.6	14.4	15.9	10.3	0.04
dnacopy.alpha ■	18.4	21.9	2.6	20.4	24.6	2.0	25.90
glad.MinBkpWeight ■	25.2	30.0	4.3	25.8	31.2	1.3	40.88
glad.default	27.4	33.3	1.2	27.4	33.3	1.2	1.13
dnacopy.prune ■	27.9	31.9	17.1	31.1	36.5	10.6	35.17
dnacopy.default	40.5	49.3	0.5	40.5	49.3	0.5	1.78
cghseg.mBIC	40.9	49.4	0.0	40.9	49.4	0.0	1.77
gada.default	80.5	96.9	0.0	80.5	96.9	0.0	0.23
cghFLasso	80.8	97.2	0.0	80.8	97.2	0.0	0.14

Table 3.3: Leave-one-out cross-validation over the 6 annotated regions was used to estimate breakpoint detection error, false positive (FP), and false negative (FN) rates. Each line shows the performance of one of the models described in the Algorithms section. Models that have a smoothness parameter are shown with a colored square, and global and local training procedures described in the Methods section were used to learn smoothness parameters. The global error is used to order the rows of the table. The Timings column shows the median time to fit the sequence of smoothing models for a single profile.

Only a few profiles need to be annotated for a good global model

Finally, to estimate the generalization error of a global model trained on a relatively small training set of t annotated profiles, we applied the n/t-fold cross-validation procedure to the data. For several training set sizes t, we plot the accuracy of the glad.lambdabreak, dnacopy.sd, flsa.norm, and cghseg.k models in Figure 3.4.

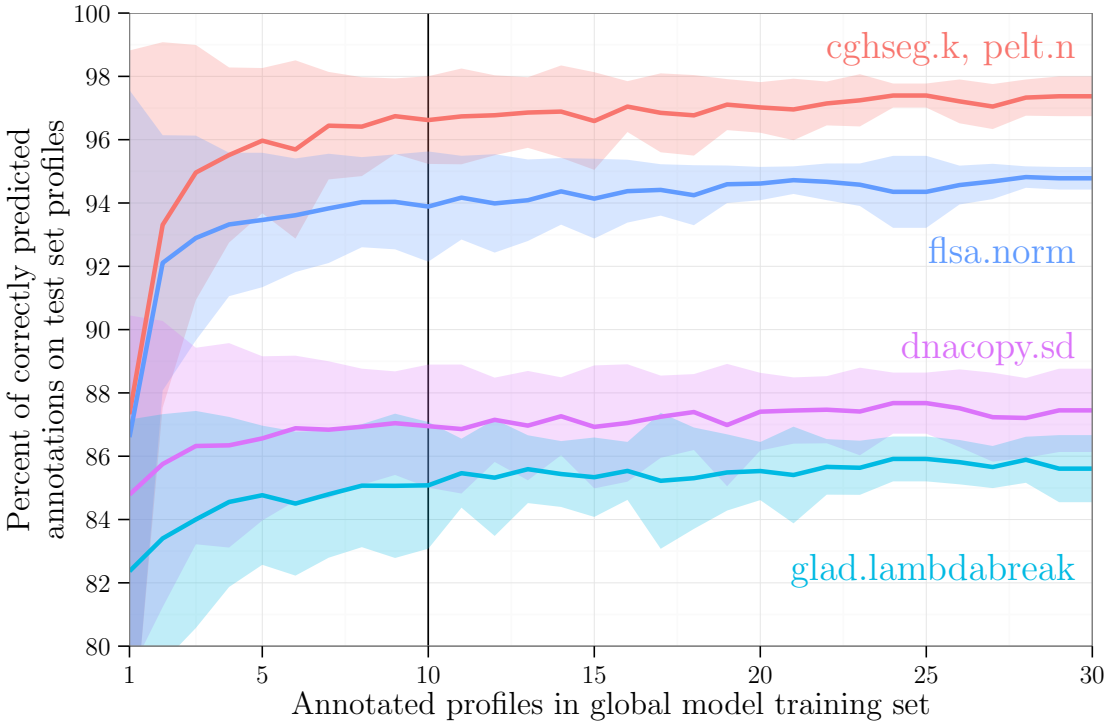

Figure 3.4: Cross-validation was used to estimate the generalization ability of the global models with different sized training sets for several breakpoint detection algorithms. For each training set size t, the profiles were partitioned into training sets of approximately size t, then were evaluated using the annotations from all the other profiles. Results on these data indicate increasing accuracy (lines) and decreasing standard deviation (shaded bands) as the training set increases, with diminishing returns after approximately $t = 10$ profiles, indicated with a vertical black line, and shown in detail for all algorithms in Table 3.4.

It is clear that adding more annotations to the training set increases the breakpoint detection accuracy in general, but at a diminishing rate. Futhermore, it is clear that the accuracy is model-dependent. So after obtaining a moderately sized database of annotations, data analysis time is better spent designing and testing better models.

This suggests the following protocol: annotate breakpoints until you see the test accuracy curves flatten out. Then, train a global model using the entire set of annotations. In Table 3.4, we used $n/10$-fold cross-validation to estimate the error rates of models trained using 10 profiles. These error estimates are slightly larger, but the model ordering is mostly unchanged, with respect to the leave-one-out cross-validated estimates of the global model error rates in Table 3.3. In particular, cghseg.k and pelt.n still show the best performance on these data, with an estimated generalization error of 3.4%.

model	errors	sd	FP	sd	FN	sd
pelt.n	3.4	1.4	1.7	1.9	11.8	8.3
cghseg.k	3.4	1.4	1.6	1.8	12.2	8.3
gada	5.9	1.8	3.0	3.0	20.4	11.9
flsa.norm	6.1	1.7	3.2	2.7	20.5	13.8
pelt.default	8.0	0.1	1.1	0.0	42.2	0.3
glad.haarseg	12.6	1.5	13.7	2.0	7.1	1.2
dnacopy.sd	13.0	1.9	6.0	4.9	47.8	24.3
flsa	13.5	1.7	8.4	5.4	38.6	30.9
glad.lambdabreak	14.9	2.0	12.7	4.7	25.7	17.3
dnacopy.alpha	19.0	1.0	22.4	1.2	2.4	0.1
glad.MinBkpWeight	26.8	1.5	31.3	2.1	4.3	3.1
glad.default	27.4	0.1	32.6	0.2	1.6	0.1
dnacopy.prune	28.5	1.6	31.7	2.4	12.8	2.9
dnacopy.default	40.5	0.1	48.5	0.2	0.7	0.0
cghseg.mBIC	40.8	0.1	49.1	0.2	0.0	0.0
gada.default	80.5	0.2	96.7	0.1	0.0	0.0
cghFLasso	80.9	0.1	97.2	0.0	0.0	0.0

Table 3.4: The n/t-fold cross-validation protocol was used to estimate error, false positive (FP), and false negative (FN) rates. Mean and standard deviation (sd) over $\lfloor n/t \rfloor = 57$ folds are shown as percents. Squares show the same colors as in the figures, and are absent for models that have no smoothness parameters. The smoothness parameter was chosen using annotations from approximately $t = 10$ profiles.

3.4 GUI implementations

To make annotation-based breakpoint detection a feasbile approach in practice on real data, I have developed free/open-source GUI software tools for creating annotation databases. I have created 2 tools: a simple Python script called `annotate_breakpoints.py`, and a web site for interactive model building called SegAnnDB.

Simple annotator program in Python

Python is a popular free/open-source software programming language, and its standard library contains Tkinter for GUI development. I used Tkinter to write `annotate_breakpoints.py`, a program consisting of 345 lines of Python code. Given copy number profiles defined in `profiles.csv` and annotations in `annotations.csv`, the following command line is used to plot the profiles and their annotations, as shown in Figure 3.5:

```
python annotate_breakpoints.py profiles.csv annotations.csv
```

The plots are interactive in the sense that they allow annotated regions to be drawn and saved for later analysis. By clicking and dragging, a red rectangle can be drawn on the plot to indicate a region that contains a breakpoint. Clicking a red rectangle changes it to a yellow rectangle, which indicates a region that contains no breakpoints. Clicking a yellow rectangle deletes that annotation. Only a few profiles from the `profiles.csv` database are displayed onscreen at a time, and so right-clicks are used to cycle the profiles. When done annotating, closing the window saves the annotations to `annotations.csv`.

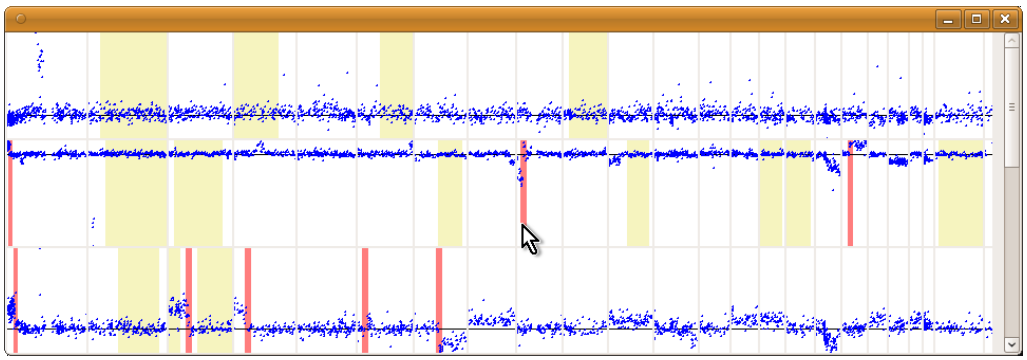

Figure 3.5: Three copy number profiles displayed by the Python GUI. Each copy number profile occupies 1 row and is divided into several panels, one for each chromosome. Blue points plot logratio as a function of chromosomal position, and breakpoint annotations can be drawn as colored rectangles.

The simple Python annotator is available in the **annotate_regions** package on the Python Package Index:

`http://pypi.python.org/pypi/annotate_regions`

SegAnnDB: a web site for supervised, interactive breakpoint detection

I implemented SegAnnDB, a web **D**ata**B**ase that can be used to **Ann**otate and **Seg**ment genomic copy number profiles. Its implementation uses approximately 4000 lines of R, Python, JavaScript, SQL, and HTML code. It works in modern web browsers that support HTML5 and Scalable Vector Graphics (SVG), so does not require installing any special software.

The web site uses SVG, HTML5, and the D3 JavaScript visualization library to display interactive plots of copy number profiles [Bostock et al., 2011]. The plots are even more interactive than the simple Python GUI, since the displayed model is updated to match the annotations drawn on the plot. After creating a breakpoint annotation, it is sent to a server which calculates which models agree. Then, the server sends the best model back to the web browser for display alongside the copy number profile. Furthermore, the scatterplots may be zoomed for identification of details in dense copy number profiles.

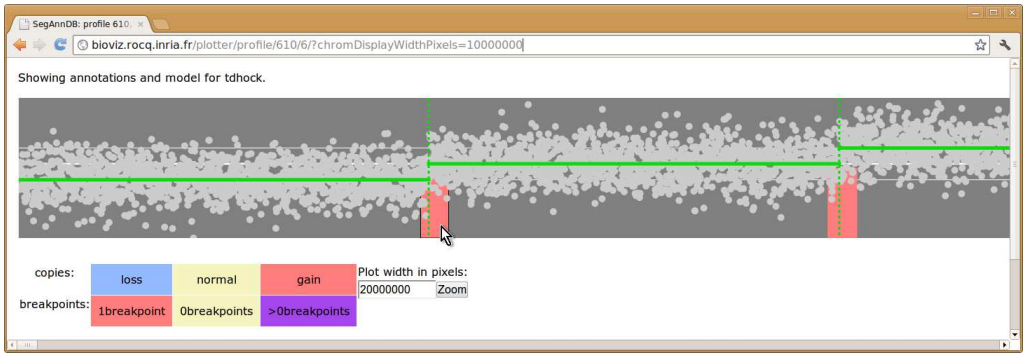

Figure 3.6: SegAnnDB chromosome detail page zoomed in. The signal in green is obtained from the grey data points using the cghseg.k model $\hat{y}^k$, with the number of segments k selected using the breakpoint annotations.

The statistical model that the server implements is a version of cghseg.k. Essentially, when a profile is uploaded to the web site, we segment it using R package **cghseg** to find the least squares segmentation for $k \in \{1, \ldots, k_{\max}\}$ segments. Then, we use Equation 3.8 to calculate the optimal number of segments $k^*(\lambda)$ for a pre-defined grid $\lambda \in \{\lambda_1, \ldots, \lambda_{100}\}$. To select which λ is best for one annotated signal, we first select only the subset of λ which maximizes agreement with that signal's breakpoint annotations. That usually leaves several tens of valid λ values for a given signal, and so then we use breakpoint annotations from other signals to decide which λ to use. This hybrid model exhibits the good training error of the local model, but also enjoys the good generalization properties of the global model.

The specific web server that I used was Apache with `mod_wsgi`, which allows Python code to control the server. I used the Django web framework with the PostgreSQL database system to store and manipulate the data, annotations, and models.

I implemented a user login system so that different people can create their own databases of annotations. This can be used to create a database that contains annotations from several different people, so in the future we can build models with user-specific breakpoint detection parameters.

Finally, I created export features so that it is easy to send segmentation results from the web site to the UCSC genome browser [Kent et al., 2002]. This facilitates visualization of copy number alteration results from an experiment alongside genomic information such as refSeq genes, as shown in Figure 3.7.

The free/open-source software that implements SegAnnDB is available in the **breakpoints** project on INRIA GForge:

`https://gforge.inria.fr/projects/breakpoints/`

Furthermore, a live web site with several real data sets already uploaded is available at:

`https://bioviz.rocq.inria.fr/plotter/`

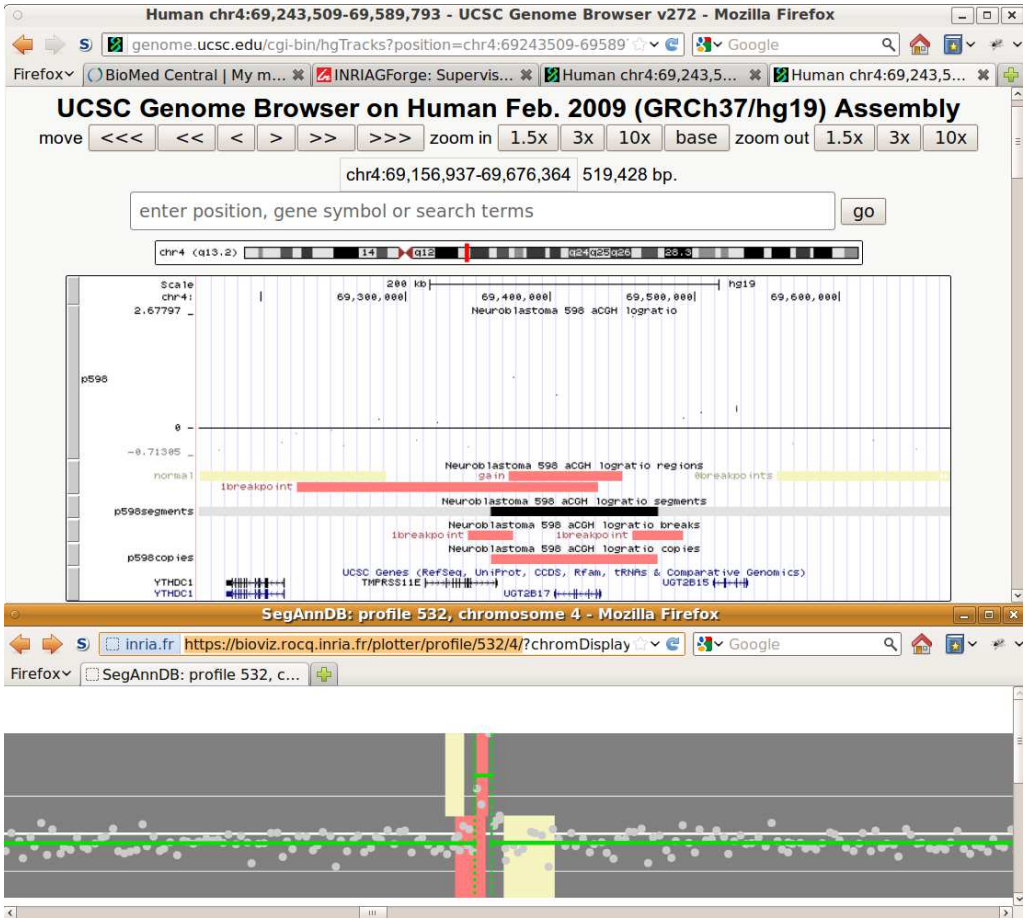

Figure 3.7: The UCSC genome browser (top window) displays data exported from SegAnnDB (bottom window). The UCSC regions track shows the breakpoint and copy number annotations using the same colors as SegAnnDB. The UCSC segments track shows the mean level of each segment using a color scale. The UCSC breaks track shows estimated breakpoint locations, drawn using dashed vertical lines on SegAnnDB. The UCSC copies track shows estimated changes in copy number, and it is clear that the displayed gain affects the copy number of the UGT2B17 gene.

3.5 Conclusions

We have proposed to train smoothing models using annotations determined by visual inspection of the copy number profiles. We have demonstrated that this approach allows quantitative comparison of smoothing models on a new data set of 575 neuroblastoma copy number profiles. These data provide the first set of annotations that can be used for benchmarking the breakpoint detection ability of future algorithms. Finally, our annotation-based approach is quite useful in practice on real data, since it provides a quantitative criterion for choosing the model and its smoothing parameter.

One possible criticism of annotation-based model selection is the time required to create the annotations. However, using the free/open-source GUIs that we have developed, it takes only a few minutes to annotate the breakpoints in a profile. This is a relatively small investment compared to the time required to write the code for data analysis, which is typically on the order of days or weeks. In addition, in the neuroblastoma data, we saw diminishing returns in breakpoint detection accuracy after annotating only 10 of 575 profiles. So breakpoint annotations are a feasible approach for finding an accurate model and smoothing parameter for real copy number profiles.

We found that global models generalize better than local models. In contrast with our results, in a previous work on automatic parameter tuning of smoothing models, Zhang et al. [2010] claim that local models should be better in some sense: "it is clear that the advantages of selecting individual-specific λ values outweigh the benefit of selecting constant λ values that maximize overall performance." It is not clear if the authors mean train or test error when they speak of "performance." Indeed, local models fit the breakpoint annotations better and thus have lower training error. However, we also showed that global models often generalize better than local models, according to our leave-one-out estimates.

We have also shown that learning a global smoothness parameter on a limited set of annotations can generalize well to un-annotated profiles. However, the smoothness parameterization must be carefully chosen. For example, the flsa.norm algorithm scales the smoothness parameter λ by the number of points and the length of the chromosome, and results in lower error rates than the unscaled flsa algorithm. We will pursue this idea in Chapter 4, when we define optimal penalties for breakpoint detection.

We have solved the problem of smoothness parameter selection using breakpoint annotations, but the biological question of detecting CNAs remains. By constructing a database of annotated regions of CNAs, we could use a similar approach to train models that detect CNAs. Annotations could

be actual copy number $(0, 1, 2, 3, \dots)$ or some simplification (loss, normal, gain). For the future, we will be interested in developing joint breakpoint detection and copy number calling models that directly use these annotation data as constraints or as part of the model likelihood.

It will be interesting to apply annotation-based model training to other algorithms and data sets. In the annotations we analyzed, cghseg.k and pelt.n showed the best breakpoint detection, but another model may be selected with another expert's annotation of the neuroblastoma data. In future work, it will be interesting to see if our conclusions are robust to the annotator, and generalize to data from other tumor types. Furthermore, since next-generation sequencing data sets are becoming more common, we plan to use visual annotations to learn segmentation models for these data as well.

Chapter 4

Designing optimal penalties for breakpoint detection using segmentation model selection

The material from this chapter has not been published elsewhere.

This chapter discusses methods for accurately detecting the breakpoints in a piecewise constant signal μ, given some noisy observations y. In the previous chapter, we saw that that models with different penalties like flsa and flsa.norm have different breakpoint detection accuracy. This observation naturally poses a question: what is the penalty that will result in the best breakpoint detection? In this chapter, I discuss a method that can be used to find these penalties, given certain assumptions about the data.

Chapter summary

This chapter presents two contributions:

- Given a latent signal μ, we define a precise breakpoint detection error function, and discuss its relationship to the annotation error defined in Chapter 3.
- We use the error function to determine optimal penalties for breakpoint detection in databases of simulated signals of varying sampling density, noise, and length.

In recent years, several authors have developed a theory of minimal penalties that can be used to accurately recover a signal from noisy observations [Arlot and Massart, 2009, Lebarbier, 2005]. These methods do not take advantage of the breakpoint annotation data, but instead can be

used offline to analyze some assumptions about the signal and the noise of the data. Typically, these results guarantee recovery of the correct signal with high probability. However, in this chapter we are more interested in accurate recovery of the breakpoints than the signal itself. So here I devote several pages to directly attacking the problem of breakpoint detection rather than signal recovery.

Starting in Section 4.7, I will explain how to construct optimal penalties given certain assumptions about the signal and noise. But we define optimality in terms of breakpoint detection error, so I will first devote several pages to a precise definition of the breakpoint detection error.

4.1 Properties of an ideal error function for breakpoint detection

We assume there is a chromosome with D base pairs. Let $\mathcal{X} = \{1, \dots, D\}$ be all the base pairs, and let $\mathbb{B} = \{1, \dots, D-1\}$ be all bases after which a break is possible.

For the simulations we explore later in this chapter, we assume there is some latent signal $\mu \in \mathbb{R}^D$. We sample some noisy signal $y \in \mathbb{R}^d$ at positions $p \in \mathcal{X}^d$, sorted in increasing order $p_1 < \cdots < p_d$. We will focus on the cghseg model, which defines the estimated signal with k segments as

$$\begin{aligned} \hat{y}^k = &\arg\min_{x \in \mathbb{R}^d} \quad ||y - x||_2^2 \\ &\text{subject to} \quad k - 1 = \sum_{j=1}^{d-1} 1_{x_j \neq x_{j+1}}. \end{aligned} \tag{4.1}$$

Note that we can quickly calculate $\hat{y}^k$ for $k \in \{1, \dots, k_{\max}\}$ using pruned dynamic programming [Rigaill, 2010]. We then estimate the breakpoint locations using the mean

$$\varphi(\hat{y}^k, p) = \left\{ \lfloor (p_j + p_{j+1})/2 \rfloor \text{ for all } j \in \{1, \dots, d-1\} \text{ such that } \hat{y}^k_j \neq \hat{y}^k_{j+1} \right\}. \tag{4.2}$$

Note that this is a function $\varphi : \mathbb{R}^d \times \mathcal{X}^d \to 2^{\mathbb{B}}$ that gives the positions after which there is a break in $\hat{y}^k$. We would like to compare these estimated breakpoints to the exact set of breakpoints in the simulated signal

$$B = \varphi\left(\mu, \begin{bmatrix} 1 & \cdots & D \end{bmatrix}'\right) = \{j \in \mathbb{B} : \mu_j \neq \mu_{j+1}\}. \tag{4.3}$$

Given some guess of the breakpoint locations $G \subseteq \mathbb{B}$, the object of this section is to define a function $E(G)$ that quantifies how bad the breakpoint location guess was. We would like the function $E : 2^{\mathbb{B}} \to \mathbb{R}^+$ to satisfy:

- **(correctness)** Guessing exactly right costs nothing: $E(B) = 0$.
- **(precision)** A guess closer to a real breakpoint is less costly: if $B = \{b\}$ and $0 \leq i < j$, then $E(\{b + i\}) \leq E(\{b + j\})$ and $E(\{b - i\}) \leq E(\{b - j\})$.
- **(FP)** False positive breakpoints are bad: if $b \in B$ and $g \notin B$, then $E(\{b\}) < E(\{b, g\})$.
- **(FN)** Undiscovered breakpoints are bad: $b \in B \Rightarrow E(\{b\}) < E(\emptyset)$.

Keeping these properties in mind, in this section we define 4 such breakpoint detection error functions E:

- When the latent signal is available in simulations, we can use the exact breakpoint locations B to define E^B_{exact}, which satisfies all 4 properties.
- In real data, we do not know the exact breakpoint locations B, but we can approximate them using regions $\hat{R}$ that visually contain breakpoints. Assuming that there is exactly 1 region in $\hat{R}$ for every breakpoint in B, we can define $E^{\hat{R}}_{\text{complete}}$, which also satisfies all 4 properties.
- In real data, we may not want to assume that the regions $\hat{R}$ contain all breakpoints B. So we define $E^{\hat{R},A}_{\text{incomplete}}$, which only satisfies these properties if the annotations A are consistent with the real breakpoints B.
- In real data, an imperfect annotator is making the database of regions $\hat{R}$ and annotations A. So we define $E^{\hat{R},A}_{01}$ to limit the influence of each annotation.

We will define each of these breakpoint detection error functions in the next sections, then compare them in Figures 4.3 and 4.4.

4.2 Exact breakpoint error for simulated signals

In this section, we use the exact breakpoint locations B to define a breakpoint detection error function.

We define the error of a breakpoint location guess $g \in \mathbb{B}$ as a function of the closest breakpoint in B. So first we put the breaks in order, by writing them as $B_1 < \cdots < B_n$, with each $B_i \in \mathbb{B}$. Then, we define a set of intervals $R_B = \{\mathbf{r}_1, \ldots, \mathbf{r}_n\}$ that form a partition of $\mathbb{B}$. For each breakpoint B_i we define the region $\mathbf{r}_i = [\underline{r}_i, \overline{r}_i] \in \mathbb{IB}$, where $\mathbb{IB} \subset 2^{\mathbb{B}}$ denotes the set of all intervals of $\mathbb{B}$. We take the notation conventions from the interval analysis literature [Nakao et al., 2010].

We define the right limit of region i as

$$\overline{r}_i = \begin{cases} D-1 & \text{if } i = n \\ \lfloor (B_{i+1} + B_i)/2 \rfloor & \text{otherwise} \end{cases} \tag{4.4}$$

and the left limit as

$$\underline{r}_i = \begin{cases} 1 & \text{if } i = 1 \\ \overline{r}_{i-1} + 1 & \text{otherwise.} \end{cases} \tag{4.5}$$

The breakpoints B_i and regions $\mathbf{r}_i$ are labeled for a small signal in Figure 4.1.

Intuitively, if we observe a breakpoint guess $g \in \mathbf{r}_i$, then its closest breakpoint is B_i. To define the best guess in each region, we use piecewise linear functions $C_{\underline{r},b,\overline{r}} : \mathbb{R} \to [0,1]$ defined as follows:

$$C_{\underline{r},b,\overline{r}}(g) = \begin{cases} 0 & \text{if } g = b \\ (b-g)/(x-\underline{r}) & \text{if } \underline{r} < g < b \\ (g-b)/(\overline{r}-x) & \text{if } b < g < \overline{r} \\ 1 & \text{otherwise.} \end{cases} \tag{4.6}$$

For each breakpoint i we measure the precision of a guess $g \in \mathbb{B}$ using

$$\ell_i(g) = C_{\underline{r}_i, B_i, \overline{r}_i}(g). \tag{4.7}$$

These functions are shown in Figure 4.1 for a small signal with 2 breakpoints.

Now, we are ready to define the exact breakpoint error of a set of guesses $G \subseteq \mathbb{B}$. First, let $G \cap \mathbf{r}$ be the subset of guesses G that fall in region $\mathbf{r}$.

Then, we define the false negative rate for region $\mathbf{r}$ as

$$\mathrm{FN}(G, \mathbf{r}) = \begin{cases} 1 & \text{if } G \cap \mathbf{r} = \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

and the false positive rate for region $\mathbf{r}$ as

$$\mathrm{FP}(G, \mathbf{r}) = \begin{cases} 0 & \text{if } G \cap \mathbf{r} = \emptyset \\ |G \cap \mathbf{r}| - 1 & \text{otherwise} \end{cases} \tag{4.9}$$

and the imprecision of the best guess in region r as

$$I(G, \mathbf{r}, \ell) = \begin{cases} 0 & \text{if } G \cap \mathbf{r} = \emptyset \\ \min_{g \in G \cap \mathbf{r}} \ell(g) & \text{otherwise.} \end{cases} \tag{4.10}$$

When there are no breakpoints, we have $B = \emptyset$ and $R_B = \emptyset$. But we still would like to quantify the false positives, so let $G \setminus \left(\cup R_B \right)$ be the set of guesses G outside of the breakpoint regions R_B. Finally, we define the exact error of guess G with respect to the true breakpoints B as

$$E^B_{\text{exact}}(G) = \left| G \setminus (\cup R_B) \right| + \sum_{i=1}^{|B|} \mathrm{FP}(G, \mathbf{r}_i) + \mathrm{FN}(G, \mathbf{r}_i) + I(G, \mathbf{r}_i, \ell_i). \tag{4.11}$$

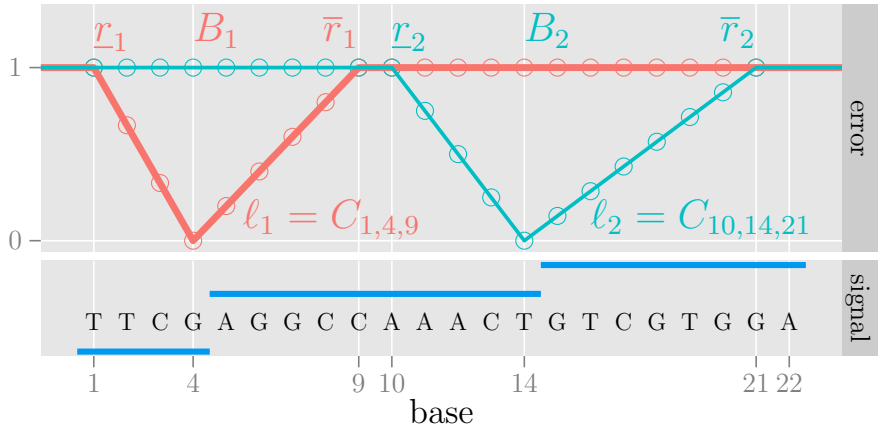

Figure 4.1: For 2 breakpoints i, we plot their functions ℓ_i that measure the precision of a guess. The signal $\mu \in \mathbb{R}^{22}$ has 2 breakpoints: $B = \{4, 14\}$.

To calculate the exact breakpoint error, we first sort lists of $n = |B|$ and $m = |G|$ items. Using the quicksort algorithm, this requires $O(n \log n + m \log m)$ operations in the average case [Cormen et al., 1990]. Once sorted, the components of the cost can be calculated in linear time $O(n + m)$. So, overall the calculation of the error can be accomplished in best case $O(n + m)$, average case $O(n \log n + m \log m)$ operations.

4.3 Incomplete annotation error for real data

In real data, we do not have access to the latent signal μ, nor the underlying set of breakpoints B. However, by plotting the data, we can easily identify regions that contain breakpoints by visual inspection, as shown in Figure 4.2.

In general, let $A = \{a_1, \ldots, a_n\}$ be some annotations and $\hat{R} = \{\hat{\mathbf{r}}_1, \ldots, \hat{\mathbf{r}}_n\}$ be the corresponding regions. For every annotation i, let $\hat{\mathbf{r}}_i \subset \mathbb{IB}$ be the interval that defines the region, and let $a_i \subseteq \{0, 1, \ldots\}$ be the interval of allowable breakpoint counts in this region. For example, consider the annotated regions in Table 4.1.

Then, we define the annotation-dependent false positive rate as

$$\hat{\mathrm{FP}}(G, \mathbf{r}, a) = \big(|G \cap \mathbf{r}| - \max(a)\big)_+ \tag{4.12}$$

and the annotation-dependent false negative rate as

$$\hat{\mathrm{FN}}(G, \mathbf{r}, a) = \big(\min(a) - |G \cap \mathbf{r}|\big)_+. \tag{4.13}$$

So then we define the incomplete annotation error for a set of regions R with annotations A as

$$E^{R,A}_{\text{incomplete}}(G) = \sum_{(\mathbf{r},a)\in(R,A)} \hat{\mathrm{FP}}(G, \mathbf{r}, a) + \hat{\mathrm{FN}}(G, \mathbf{r}, a). \tag{4.14}$$

i	Annotation a_i	Region $\hat{\mathbf{r}}_i$
1	{0}	[5,10]
2	{1}	[20,30]
3	{1,2,...}	[40,70]
4	{0}	[80,100]

Table 4.1: Sample annotated regions for a signal sampled on $D = 100$ base pairs. An annotation a_i indicates how many breakpoints are allowed in the corresponding region r_i. There are 0 breaks in bases 5-10 and 80-100. There is exactly 1 break in bases 20-30. There is at least 1 break in bases 40-70.

In the case of analyzing the simulated signals in the top panels of Figure 4.2, let us consider the set of regions $\hat{R}$ depicted using the red rectangles. Every region $\hat{\mathbf{r}}_i \in \hat{R}$ contains exactly 1 breakpoint $B_i \in \hat{\mathbf{r}}_i$, so we have a corresponding set of annotations A with $a = \{1\}$ for every annotation $a \in A$. In real data we will probably only be able to see a subset of the real breakpoints, but we analyze it here to illustrate the approximation induced by the annotation process.

Given the set of breakpoint regions $\hat{R}$, we define $|\hat{R}|+1$ negative regions

$$\hat{R}^0 = \big\{[1, \underline{r}_1 - 1], [\overline{r}_1 + 1, \underline{r}_2 - 1], \ldots, [\overline{r}_{n-1} + 1, \underline{r}_n - 1], [\underline{r}_n + 1, d - 1]\big\}, \quad (4.15)$$

as shown in the middle panels of Figure 4.2. There is a corresponding set of annotations A^0 with $a = \{0\}$ for every annotation $a \in A^0$. We will use the complete set of regions $\hat{R} \cup \hat{R}^0$ and annotations $A \cup A^0$ to define the annotation error $E^{\hat{R}\cup\hat{R}^0, A\cup A^0}_{\text{incomplete}}$ for models of these simulated signals.

In Figure 4.3, we plot some model selection error functions for the 2 simulated signals shown in Figure 4.2. It is clear that the annotation error is a good approximation of the breakpoint error, and there are several interesting observations to note.

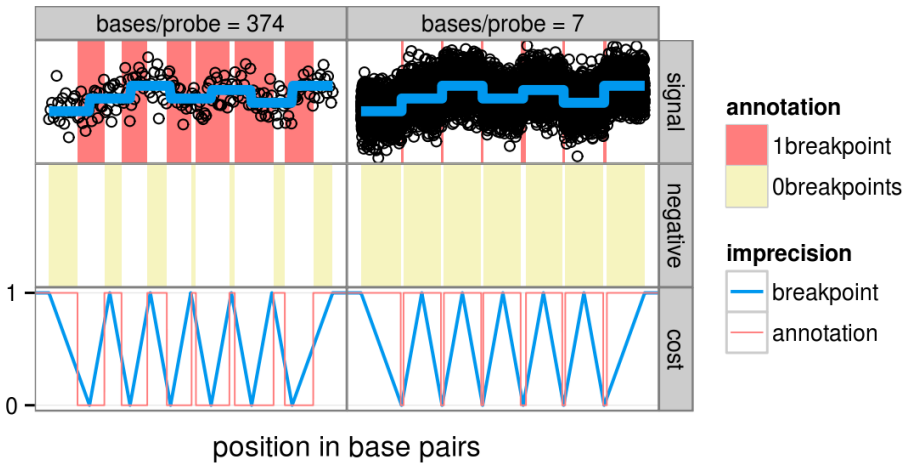

Figure 4.2: **Top**: simulated noisy signals (black) with their latent signals μ (blue) and visually-determined breakpoint annotations $\hat{R}$ (red).
Middle: negative regions $\hat{R}^0$ constructed using (4.15).
Bottom: breakpoint detection imprecision curves for the breakpoint error ℓ_i (4.10) and the annotation error $\hat{\ell}_i$ (4.17).

- **Signal**: we plot the log squared error of the estimated signal $\hat{y}^k$ with respect to the latent signal μ, defined as

$$E^{\text{signal}}(k) = \log_{10}\left[\frac{1}{d}\sum_{i=1}^{d}(\hat{y}_i^k - \mu_i)^2\right]. \tag{4.16}$$

  - For the signal sampled at 7 bases/probe, the minimum of the error identifies the correct model with 7 segments.
  - For the signal sampled at 374 bases/probe, the minimum of the error identifies an incorrect model with only 5 segments.

- **Breakpoint**: for both signals, the minimum of the breakpoint error identifies the correct model.

- **Annotation**: the annotation error is a close approximation of the breakpoint error, and also identfies the correct model.

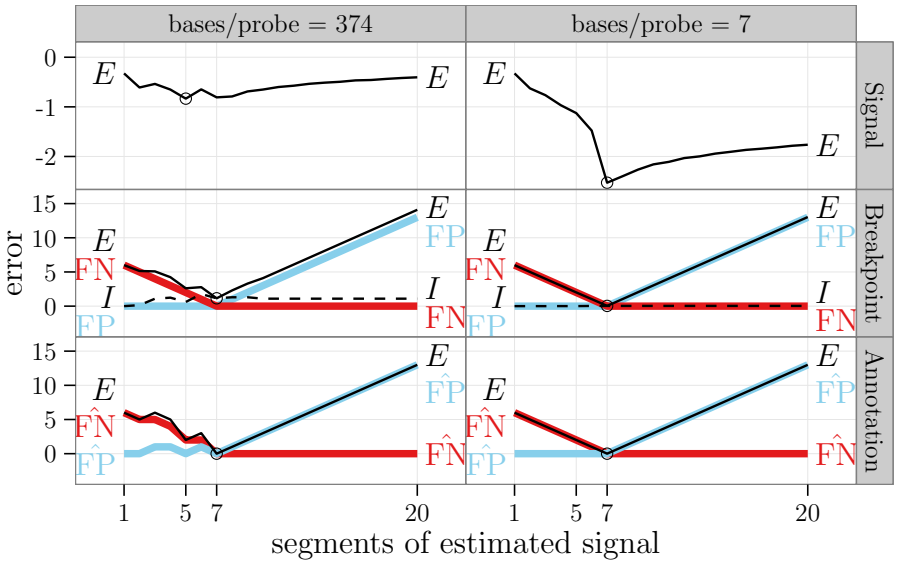

Figure 4.3: Model selection error curves for 2 simulated signals. Minima are highlighted using circles.
Signal: the log squared error E^{signal} of the estimated signal $\hat{y}^k$ for k segments with respect to the latent signal μ.
Breakpoint: exact breakpoint error E^{B}_{exact}.
Annotation: incomplete annotation error $E^{R,A}_{\text{incomplete}}$.

4.4 Link with breakpoint error using complete annotation error

It is clear from Figure 4.3 that the annotation error is a good approximation of the exact breakpoint error when the annotated regions $\hat{R}, A$ agree with the real breakpoints B. In this section, we make this intuition precise by showing exactly how to relax the breakpoint error to obtain the annotation error. There are two steps:

- First, we define the complete annotation error by relaxing the definition of the exact breakpoint error.
- Then, we show that the complete annotation error is equivalent to the incomplete annotation error when we have a complete set of annotations.

The complete annotation error

We define the complete annotation error as a relaxation of the exact breakpoint error. So first, let us recall the definition of the exact breakpoint error from Equation 4.11:

$$E^B_{\text{exact}}(G) = \left|G \setminus (\cup R_B)\right| + \sum_{i=1}^{|B|} \text{FP}(G, \mathbf{r}_i) + \text{FN}(G, \mathbf{r}_i) + I(G, \mathbf{r}_i, \ell_i).$$

To define the complete annotation error, we perform two relaxations:

- Instead of using Equations 4.4 and 4.5 to determine a breakpoint region $\mathbf{r}_i$, we use the region $\hat{\mathbf{r}}_i$ determined by visual inspection.
- Rather than the piecewise linear imprecision ℓ_i, we use the zero-one imprecision $\hat{\ell}_i$:
$$\hat{\ell}_i(g) = 1_{g \notin \hat{\mathbf{r}}_i}. \tag{4.17}$$
We show this relaxation by ploting the imprecision functions ℓ_i and $\hat{\ell}_i$ in the bottom panels of Figure 4.2.

So, performing these two relaxations results in the following definition of the complete annotation error:

$$\begin{aligned} E^{\hat{R}}_{\text{complete}}(G) &= \left|G \setminus (\cup \hat{R})\right| + \sum_{i=1}^{|\hat{R}|} \text{FP}(G, \hat{\mathbf{r}}_i) + \text{FN}(G, \hat{\mathbf{r}}_i) + I(G, \hat{\mathbf{r}}_i, \hat{\ell}_i) \\ &= \left|G \setminus (\cup \hat{R})\right| + \sum_{\hat{\mathbf{r}} \in \hat{R}} \text{FP}(G, \hat{\mathbf{r}}) + \text{FN}(G, \hat{\mathbf{r}}) \end{aligned} \tag{4.18}$$

It is clear that $E^{\hat{R}}_{\text{complete}}$ depends on the annotations only through their regions. In particular, the annotated breakpoint counts $a_i = \{1\}$ are not used in this definition, since we assumed that each region $\hat{\mathbf{r}}_i$ contains exactly 1 break. Also, since we used the zero-one loss for $\hat{\ell}_i$, the imprecision function I is always zero.

Equivalence of complete and incomplete annotation error

To see the connection between the complete and incomplete annotation error functions, note that

$$\begin{aligned} \hat{\text{FN}}(G, \mathbf{r}, \{1\}) &= \big(1 - |G \cap \mathbf{r}|\big)_+ \\ &= \text{FN}(G, \mathbf{r}), \end{aligned} \tag{4.19}$$

and

$$\begin{aligned} \hat{\text{FP}}(G, \mathbf{r}, \{1\}) &= \big(|G \cap \mathbf{r}| - 1\big)_+ \\ &= \text{FP}(G, \mathbf{r}). \end{aligned} \tag{4.20}$$

For the complete annotation error we quantified the false positive rate of the breakpoints that fall outside of the breakpoint regions $\hat{R}$ using $G \setminus (\cup \hat{R})$. For the incomplete annotation error, we instead created a set of 0-breakpoint annotations $\hat{R}^0$ for this purpose. Note that by construction of the negative regions in Equation 4.15, we have

$$G \setminus \left(\cup \hat{R}\right) = G \cap \left(\cup \hat{R}^0\right), \tag{4.21}$$

or in words, the guesses outside of the breakpoint regions $\hat{R}$ are in the negative regions $\hat{R}^0$. So using Equation 4.21, we have

$$\begin{aligned} \hat{\text{FP}}(G, (\cup \hat{R}^0), \{0\}) &= |G \cap (\cup \hat{R}^0)| \\ &= |G \setminus (\cup \hat{R})|, \end{aligned} \tag{4.22}$$

which is the first component of the complete annotation loss.

Recall that $\hat{R}, A$ represent annotated regions that each contain exactly 1 breakpoint, and $\hat{R}^0, A^0$ contain no breakpoints. So using Equations 4.19, 4.20, and 4.22, we can show that the incomplete annotation error is equiv-

alent to the complete error in this sense:

$$\begin{aligned} E^{\hat{R}\cup\hat{R}^0,A\cup A^0}_{\text{incomplete}}(G) &= \sum_{\mathbf{r}\in\hat{R}^0} \hat{\text{FP}}(G,\mathbf{r},\{0\}) + \sum_{\mathbf{r}\in\hat{R}} \hat{\text{FP}}(G,\mathbf{r},\{1\}) + \hat{\text{FN}}(G,\mathbf{r},\{1\}) \\ &= \hat{\text{FP}}(G,\cup\hat{R}^0,\{0\}) + \sum_{\mathbf{r}\in\hat{R}} \hat{\text{FP}}(G,\mathbf{r},\{1\}) + \hat{\text{FN}}(G,\mathbf{r},\{1\}) \\ &= |G\setminus(\cup\hat{R})| + \sum_{\mathbf{r}\in\hat{R}} \text{FP}(G,\mathbf{r}) + \text{FN}(G,\mathbf{r}) \\ &= E^{\hat{R}}_{\text{complete}}(G). \end{aligned} \tag{4.23}$$

So in fact the incomplete annotation error is equivalent to the complete error when the annotated regions $\hat{R}$ each contain exactly 1 breakpoint. But we call this the incomplete error since it is also well-defined for arbitrary sets of regions $\hat{R}$ and annotations A.

4.5 Zero-one annotation error

The initial method that we used to quantify the breakpoint error on the neuroblastoma data set in Chapter 3 was the zero-one loss in Equation 3.4. In this section, we show that the zero-one loss is a thresholded version of the incomplete annotation error function.

First, let us define the zero-one thresholding function $t:\mathbb{Z}^+\to\mathbb{Z}^+$ as

$$t(x) = 1_{x\neq 0} = \begin{cases} 1 & \text{if } x\neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.24}$$

The idea of thresholding is to limit the error that any one annotation can induce. So instead of counting incorrect breakpoint guesses, we count incorrect annotations. We define the zero-one annotation error as

$$\begin{aligned} E^{\hat{R},A}_{01}(G) &= \sum_{(\mathbf{r},a)\in(\hat{R},A)} t\left[\hat{\text{FP}}(G,\mathbf{r},a)\right] + t\left[\hat{\text{FN}}(G,\mathbf{r},a)\right] \\ &= \sum_{(\mathbf{r},a)\in(\hat{R},A)} 1_{|G\cap\mathbf{r}|>\max(a)} + 1_{|G\cap\mathbf{r}|<\min(a)} \\ &= \sum_{(\mathbf{r},a)\in(\hat{R},A)} 1_{|G\cap\mathbf{r}|\notin a}. \end{aligned} \tag{4.25}$$

4.6 Comparing annotation error functions

In practice, we have few annotated regions per signal in real data. In Figure 4.4, we show how the annotation error is degraded as we remove annotations. In particular, it is clear that using the thresholded zero-one annotation error significantly degrades the approximation of the FP curve. Nevertheless, it is worth noting that minimum of the zero-one error still uniquely identifies the correct model with 7 segments. Even after removing many annotations, the minimum error still identifies the correct model, but not uniquely.

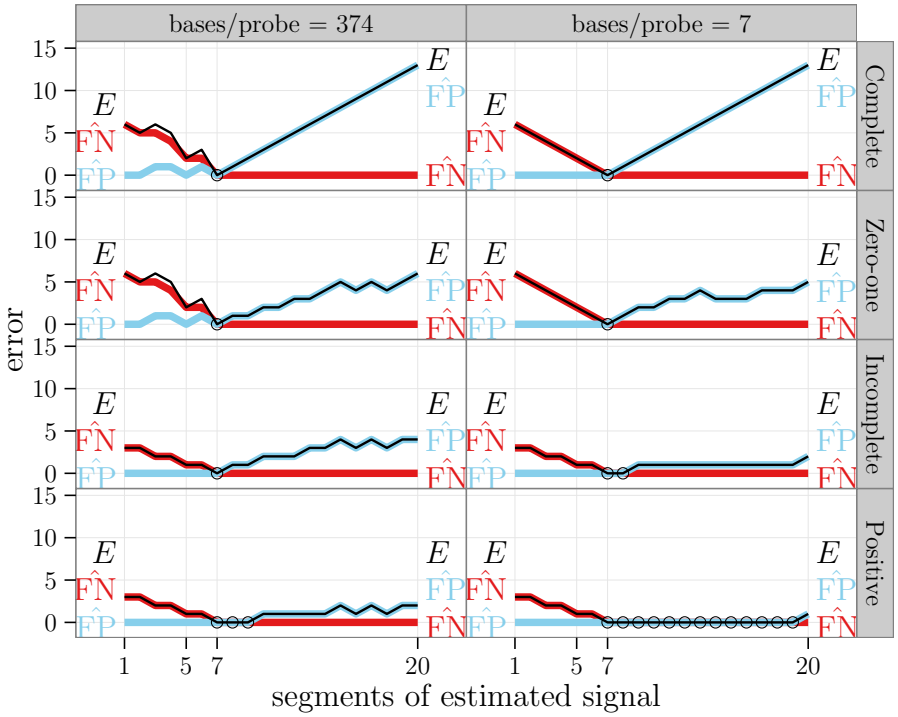

Figure 4.4: Comparison of annotation error functions as the set of annotations changes. Minima are highlighted using circles.
Complete: annotation error $E^{\hat{R}\cup\hat{R}^0,A\cup A^0}_{\text{incomplete}}$ for a complete set of 6 positive and 7 negative annotations.
Zero-one: zero-one annotation error $E^{\hat{R}\cup\hat{R}^0,A\cup A^0}_{01}$ for the complete annotations.
Incomplete: zero-one annotation error $E^{\hat{R},A}_{01}$ for 3 positive and 4 negative annotations.
Positive: zero-one annotation error $E^{\hat{R},A}_{01}$ for 3 positive annotations.

4.7 Sampling density normalization

Having properly defined how to compute the breakpoint detection error in the previous sections, we now use it to derive several results about optimal penalties for breakpoint detection. We start by considering a penalty that is invariant to sampling density. First, we will present an empirical analysis of several simulated signals using the breakpoint error. Then, we will discuss the relationship of our results to relevant theoretical results.

In real array CGH data, the sampling density of probes along the genome is not uniform across samples. In fact, we see a sampling density between 40 and 4400 kilobases per probe in the neuroblastoma data set.

So to construct a penalty that can best adapt to this variation, we analyze the following simulation. We create a latent signal $\mu \in \mathbb{R}^D$ over $D = 70000$ base pairs, with breakpoints every 10000 base pairs, shown as the blue line in Figure 4.5. Then, we define a signal sample size $d_i \in \{70, \ldots, 70000\}$ for every noisy signal $i \in \{1, \ldots, n\}$. Let $y_i \in \mathbb{R}^{d_i}$ be noisy signal i, sampled at positions $p_i \in \mathcal{X}^{d_i}$, with $p_{i1} < \cdots < p_{i,d_i}$. We sample every probe j from the $y_{ij} \sim N(\mu_{p_{ij}}, 1)$ distribution. These samples are shown as the black points in Figure 4.5.

We would like to learn some model complexity parameter λ on the first noisy signal, and use it for accurate breakpoint detection on the second noisy signal. In other words, we are looking for a model selection criterion which is invariant to sampling density.

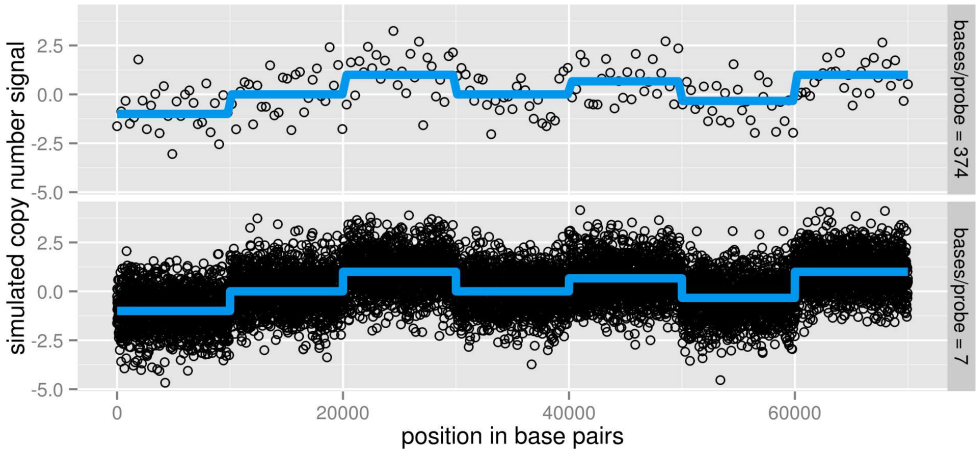

Figure 4.5: Two noisy signals (black) sampled from a latent piecewise constant signal (blue). Note that these are the same signals that appear in Figure 4.2.

To attack this problem, we proceed as follows. For every signal i, we use pruned dynamic programming to calculate the maximum likelihood estimator $\hat{y}_i^k \in \mathbb{R}^{d_i}$, for several model sizes $k \in \{1, \ldots, k_{\max}\}$ [Rigaill, 2010]. Then, we define the model selection criteria

$$k_i^\alpha(\lambda) = \arg\min_k \lambda k d_i^\alpha + ||y_i - \hat{y}_i^k||_2^2. \tag{4.26}$$

Each of these is a function $k_i^\alpha : \mathbb{R}^+ \to \{1, \ldots, k_{\max}\}$ that takes a model complexity tradeoff parameter λ and returns the optimal number of segments for signal i. The goal is to find a penalty exponent $\alpha \in \mathbb{R}$ that lets us generalize λ between different signals i.

To quantify the accuracy of a segmentation for signal i, let $\text{BErr}_i(k)$ be the breakpoint detection error of the model with k segments. This is a function $\text{BErr}_i : \{1, \ldots, k_{\max}\} \to \mathbb{R}^+$, and in real data this corresponds to the breakpoint annotation error $E_{01}^{\hat{R},A}\left[\varphi(\hat{y}_i^k, p_i)\right]$. But in these simulations we can calculate the more precise measure

$$\text{BErr}_i(k) = E_{\text{exact}}^{B}\left[\varphi(\hat{y}_i^k, p_i)\right]. \tag{4.27}$$

where B is the set of real breakpoints in the latent signal μ.

In Figure 4.6, we plot BErr_i for the 2 simulated signals i shown previously. As expected, the model recovers more accurate breakpoints from the signal sampled at a higher density.

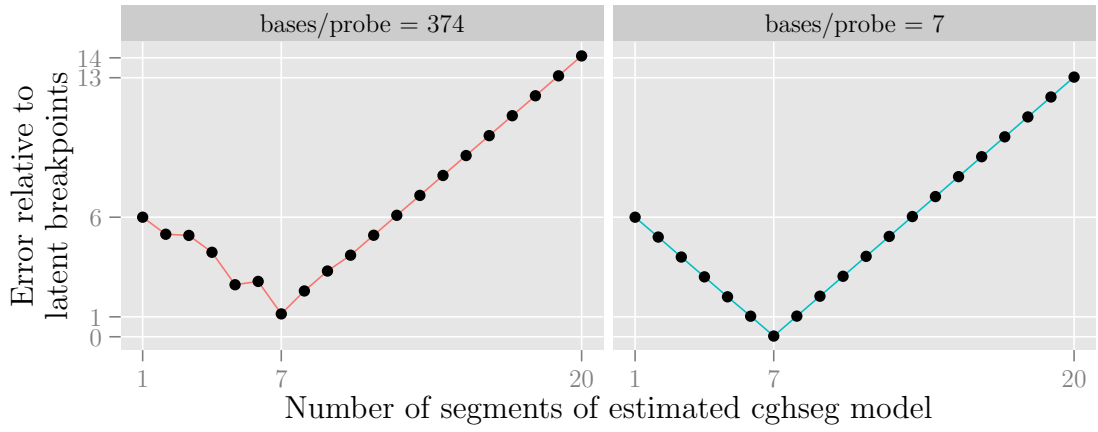

Figure 4.6: Exact breakpoint error $\text{BErr}_i(k)$ for two signals i and several cghseg model sizes k. Note that these are the same error curves that appear in the Breakpoint panels of Figure 4.3.

Now, let us define the penalized model breakpoint error $E_i^\alpha : \mathbb{R}^+ \to \mathbb{R}^+$ as

$$E_i^\alpha(\lambda) = \text{BErr}_i\left[k_i^\alpha(\lambda)\right]. \tag{4.28}$$

In Figure 4.7, we plot these functions for the two signals i shown previously, and for several penalty exponents α.

The dots in Figure 4.7 show the optimal λ found by minimizing the penalized model breakpoint detection error:

$$\hat{\lambda}_i^\alpha = \arg\min_{\lambda \in \mathbb{R}^+} E_i^\alpha(\lambda) \tag{4.29}$$

Figure 4.7 suggests that $\alpha \approx 1/2$ defines a penalty with aligned error curves, which will result in $\hat{\lambda}_i^\alpha$ values that can be generalized between profiles.

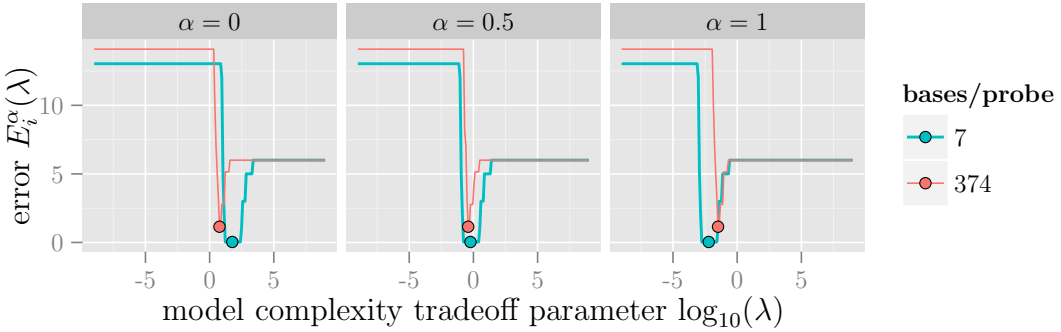

Figure 4.7: Model selection error curves $E_i^\alpha(\lambda)$ for 2 signals i and several exponents α. The penalty contains a term for the number of points sampled d_i^α.

Now, we are ready to define 2 quantities that will be able to help us choose an optimal penalty exponent α.

First, let us consider the training error over the entire database:

$$E^{\alpha}(\lambda) = \sum_{i=1}^{n} E_i^{\alpha}(\lambda), \tag{4.30}$$

and we define the minimal value of this function as

$$E^{*}(\alpha) = \min_{\lambda} E^{\alpha}(\lambda). \tag{4.31}$$

In Figure 4.8, we plot these training error functions E^{α} and their minimal values E^{*} for several values of α. It is clear that the minimum training error is found for some penalty exponent α near $1/2$, and we would like to find the precise α that results in the lowest possible minimum $E^{*}(\alpha)$.

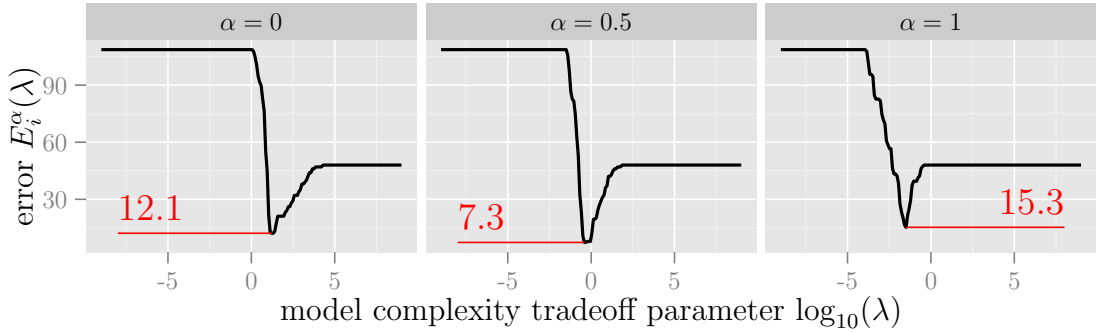

Figure 4.8: Training error functions E^{α} in black and their minimal values $E^{*}(\alpha)$ in red. The penalty contains a term for the number of points sampled d_i^{α}.

We also consider the test error over all pairs of signals when training on one and testing on another:

$$\text{TestErr}(\alpha) = \sum_{i \neq j} E_i^{\alpha}(\hat{\lambda}_j^{\alpha}). \tag{4.32}$$

In Figure 4.9, we plot E^* and TestErr for a grid of α values. It is clear that the optimal penalty is given by $\alpha = 1/2$. This corresponds to the following model selection criterion which is invariant to sampling density:

$$k_i(\lambda) = \arg\min_k \lambda k \sqrt{d_i} + ||y_i - \hat{y}_i^k||_2^2 \tag{4.33}$$

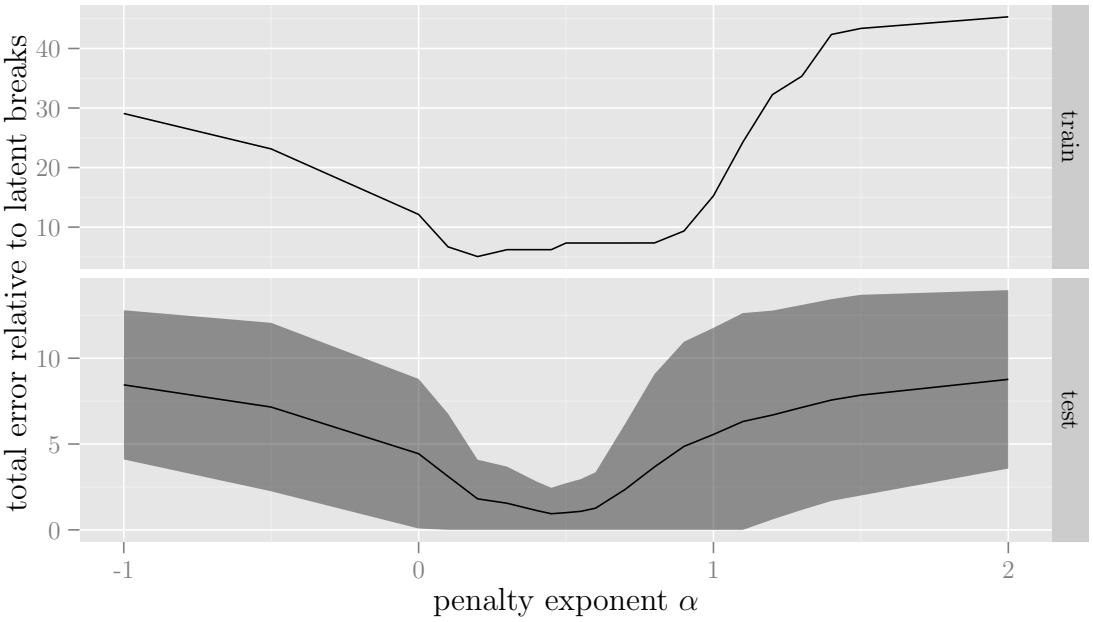

Figure 4.9: Train and test breakpoint detection error as a function of penalty exponent α. The penalty contains a term for the number of points sampled d_i^{α}. Mean error is drawn as a black line, with one standard deviation shown as a grey band.

As explained by Arlot and Celisse [2010], a model selection procedure can be either efficient or consistent. An efficient procedure for model estimation accurately recovers the latent signal, whereas a consistent procedure for model identification accurately recovers the breakpoints. Since we consider the breakpoint detection error, we are attempting to construct a consistent penalty, not an efficient penalty.

In general terms, the fact that we find a nonzero exponent α for our d_i^α penalty term agrees with other results. In particular, Arlot [2008] proposed an optimal procedure to select model complexity parameters in cross-validation by normalizing by the sample size d_i.

The $\sqrt{d_i}$ term that we find here using simulations is in agreement with Fischer [2011], who used finite sample model selection theory to find a $\sqrt{d_i}$ term in a penalty optimal for clustering.

When theoretically deriving an efficient penalty for change-point model estimation in the non-asymptotic setting, Lebarbier [2005] obtained a $\log d_i$ term. This contrasts our result, which examines the identification problem using the breakpoint error and obtains a $\sqrt{d_i}$ term. But in fact this is in agreement with classical results that AIC underpenalizes with respect to the BIC, as shown in Table 4.2.

Estimation Model	Penalty Term	Identification Model	Penalty Term
AIC	2	BIC	$\log d_i$
Lebarbier	$\log d_i$	This work	$\sqrt{d_i}$

Table 4.2: Comparing our results with Lebarbier, in the context of classical results involving AIC and BIC. The BIC is designed for model identification and penalizes more than the AIC. Likewise, our penalty examines model identification using the breakpoint detection error, and penalizes more than the efficient penalty proposed by Lebarbier.

4.8 Scale normalization

In real array CGH data, the signal variance is not the same across samples. In fact, using the variance estimate proposed below in Equation 4.34, over all the profiles in the neuroblastoma data set, we see a range of values between 0.029 and 0.371.

So we analyze the following simulation to construct a penalty that is invariant to the scale of the data. First, we define a theoretical signal $\mu \in \mathbb{R}^{700}$ with breakpoints every 100 bases, shown as the blue line for 2 signals in Figure 4.10.

Then, we generate a signal $y_i \in \mathbb{R}^{d_i}$ by adding standard normal noise to the signal, and multiplying by a scale factor $\sigma \in \{1, 10, 100, 1000\}$. The goal is to define a penalty invariant to this scaling. For each signal i, we estimate its variance using the robust estimator

$$\hat{s}_i = \text{Median}_{j=1}^{d_i-1} \left(|y_{ij} - y_{i,j+1}| \right), \tag{4.34}$$

and then define the optimal number of segments as

$$k_i^\alpha(\lambda) = \arg\min_k \lambda k \hat{s}_i^\alpha + ||y_i - \hat{y}_i^k||_2^2, \tag{4.35}$$

where $\hat{y}_i^k$ is the least squares segmentation with k segments.

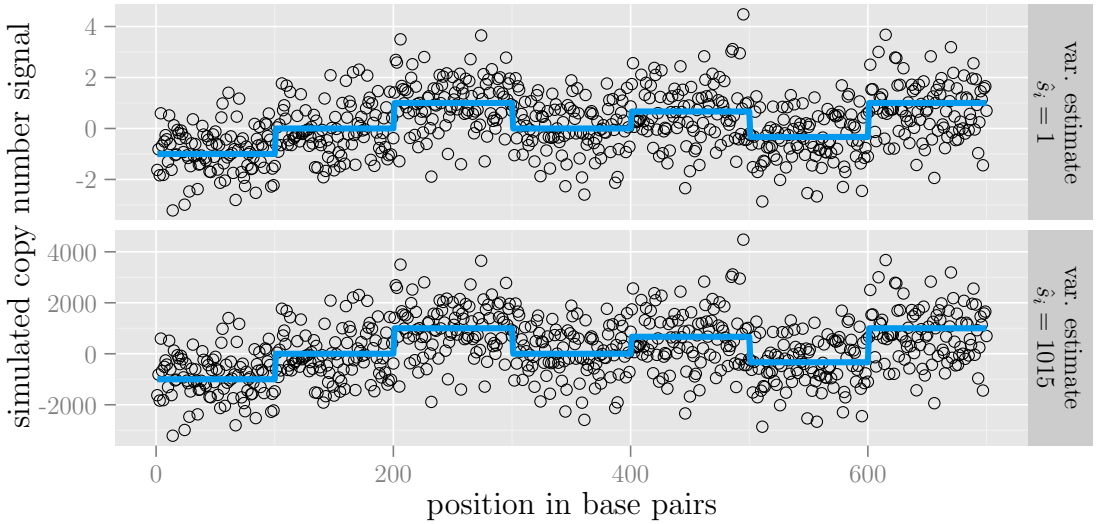

Figure 4.10: Simulated noisy signal (black) and latent signal (blue) for 2 different scales.

Using the type of analysis in the previous section, we saw that minimum breakpoint detection error is achieved when we choose a penalty exponent α that results in aligned annotation error curves E_i^α. For two signals $i \neq j$, this implies the condition

$$\text{BErr}_i[k_i^\alpha(\lambda)] \quad = \quad \text{BErr}_j[k_j^\alpha(\lambda)]. \tag{4.36}$$

Since the assumption of our simulation model is that signals i and j are identical up to a scaling, the breakpoints are the same and so $\text{BErr}_i = \text{BErr}_j$. Thus we simply need to find the exponent α such that $k_i^\alpha(\lambda) = k_j^\alpha(\lambda)$.

It is easy to see why $\alpha = 2$ is the optimal exponent. First, let y_1, y_2 be two signals which are equivalent up to a scaling factor: $y_2 = \sigma y_1$. This implies two interesting properties:

- **(a)** estimated models are the same up to a scaling factor: $\hat{y}_2^k = \hat{y}_1^k \sigma$ for all model sizes k.
- **(b)** variance estimates are the same up to a scaling factor: $\hat{s}_2 = \hat{s}_1 \sigma$.

From the defintion of k_i^α in Equation 4.35 it is clear that

$$\begin{aligned}
k_2^0(\lambda) &= \arg\min_k \lambda k + ||y_2 - \hat{y}_2^k||_2^2 \\
&= \arg\min_k \lambda k + ||\sigma y_1 - \sigma \hat{y}_1^k||_2^2 \quad \textbf{(a)} \\
&= \arg\min_k \lambda k + \sigma^2 ||y_1 - \hat{y}_1^k||_2^2 \\
&= \arg\min_k \lambda \sigma^{-2} k + \sigma^2 ||y_1 - \hat{y}_1^k||_2^2 \\
&= k_1^0(\lambda \sigma^{-2}).
\end{aligned}$$

We use this fact to rewrite the equality of the model selection curves as follows:

$$\begin{aligned}
k_1^\alpha(\lambda) &= k_2^\alpha(\lambda) \\
k_1^0(\lambda \hat{s}_1^\alpha) &= k_2^0(\lambda \hat{s}_2^\alpha) \\
&= k_1^0(\lambda \hat{s}_2^\alpha \sigma^{-2}) \\
&= k_1^0\big(\lambda (\hat{s}_1 \sigma^\alpha) \sigma^{-2}\big) \quad \textbf{(b)} \\
&= k_1^0(\lambda \hat{s}_1^\alpha \sigma^{\alpha-2}).
\end{aligned}$$

This implies that $\alpha = 2$ must be chosen in order for the model selection curves to align $k_1^\alpha(\lambda) = k_2^\alpha(\lambda)$.

We now proceed with the same kind of empirical analysis that we used to establish an optimal penalty for sampling density. In Figure 4.11, we plot the penalized model breakpoint detection error E_i^α and the optimal $\hat{\lambda}_i^\alpha$ for two signals i and several α values. This plot suggests that a value of $\alpha = 2$ defines a penalty that will allow generalization of $\hat{\lambda}_i^\alpha$ values between signals i.

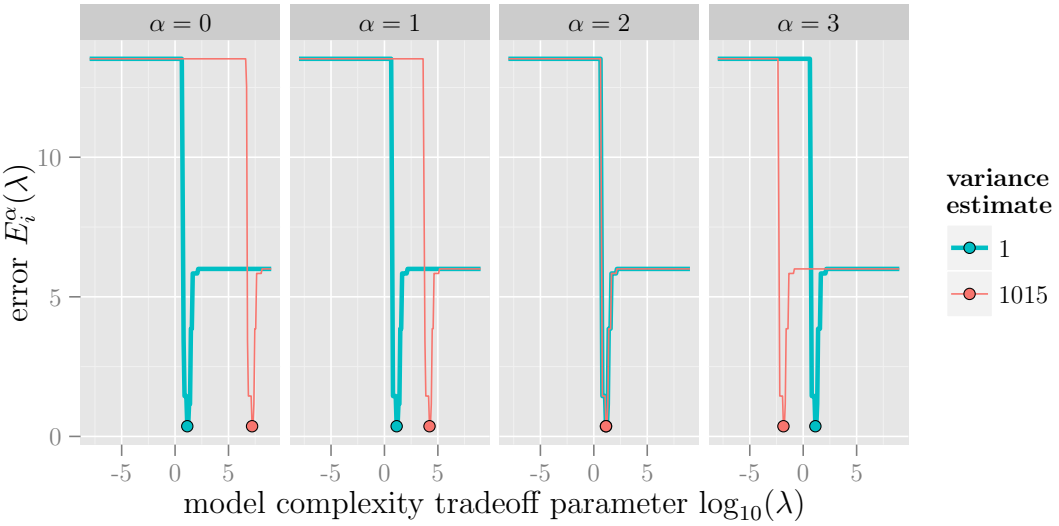

Figure 4.11: Model breakpoint detection error functions E_i^α (lines) and optimal $\hat{\lambda}_i^\alpha$ (points) for several penalty exponents α and 2 signals i of different scale. The penalty contains a term for the variance estimate $\hat{s}_i^\alpha$.

In Figure 4.12, we plot the train error E^* and TestErr as functions of α. In agreement with our earlier analysis, these simulations clearly indicate that $\alpha = 2$ is optimal. Thus, we conclude that the following model selection criterion is invariant to scaling:

$$k_i(\lambda) = \arg\min_k \lambda k \hat{s}_i^2 + ||y_i - \hat{y}_i^k||_2^2. \tag{4.37}$$

This result agrees with the theoretical results of Lebarbier [2005], who considered constructing an optimal penalty for change-point detection with respect to the square loss.

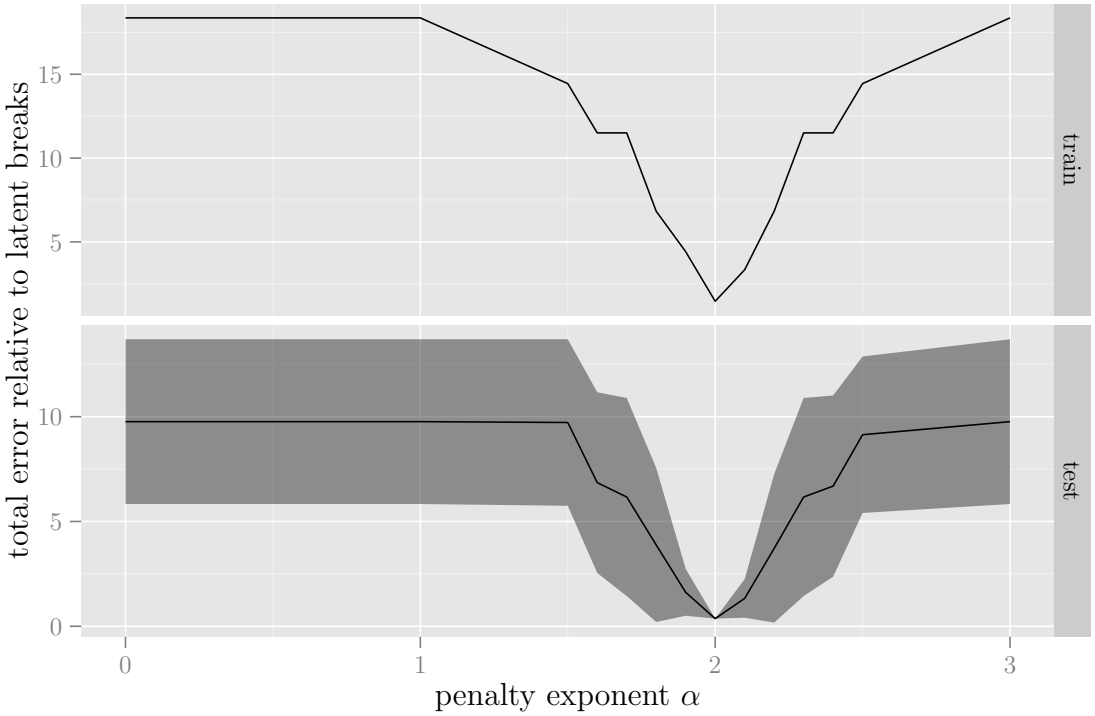

Figure 4.12: Train and test error curves as a function of penalty exponent α, for signals of variable scale. The penalty contains a term for the variance estimate $\hat{s}_i^\alpha$.

4.9 Signal length normalization

In real array CGH data, we need to analyze chromosomes of varying length in base pairs. For example, human chromosome 1 is the largest at about 250 mega base pairs, and chromosome 22 is the smallest with only about 36 mega base pairs. But we expect that the number of breakpoints is proportional to the length of the chromosome in base pairs, and we would like to design a model selection criterion that is invariant to the signal length.

So we consider the following simulation where we fix the number of points sampled at $d = 2000$ and vary the length of the signal sampled. In Figure 4.13, we show samples of 2 different lengths l_i, for the same latent signal μ.

Figure 4.13: Samples of 2 different lengths l_i but constant number of points $d = 2000$.

For each signal i, we define the penalty

$$k_i^\alpha(\lambda) = \arg\min_k \lambda k l_i^\alpha + ||y_i - \hat{y}_i^k||_2^2, \tag{4.38}$$

where l_i is the length of the signal in base pairs. The goal will be to find an α that can be used for signals of varying length.

In Figure 4.14, we show the breakpoint detection error curves for two signals and several penalty exponents α. These curves seem to align when $\alpha = -1/2$.

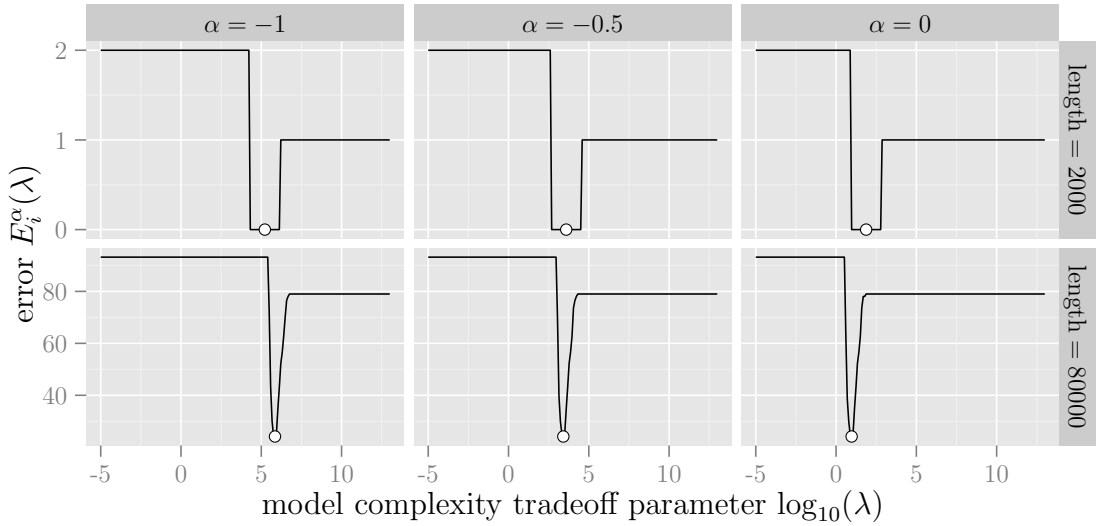

Figure 4.14: Breakpoint detection error curves for several penalty exponents α and 2 samples of varying length in base pairs l_i. The penalty contains a term l_i^α.

In Figure 4.15, we plot the train and test error curves over the entire set of simulated signals. These curves indicate minimal breakpoint detection error at $\alpha = -1/2$, corresponding to the following penalty:

$$k_i(\lambda) = \arg\min_k \frac{\lambda k}{\sqrt{l_i}} + ||y_i - \hat{y}_i^k||_2^2. \tag{4.39}$$

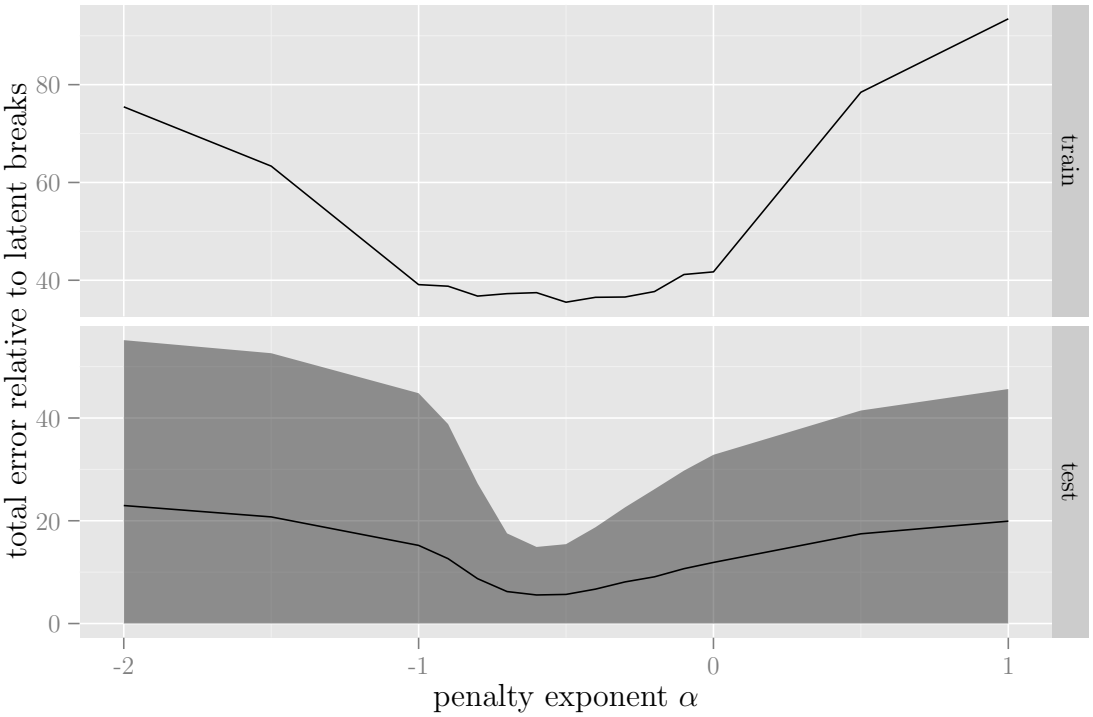

Figure 4.15: Train and test error curves for signals of different length in base pairs l_i. The penalty contains a term l_i^α.

Interestingly, the $1/\sqrt{l_i}$ term that we obtain here is in good agreement with our previous result that the optimal penalty for variable sampling density d_i should have a $\sqrt{d_i}$ term. In particular, we can re-parameterize the problem to be in terms of the number of points sampled per segment $\rho_i = d_i/k_i$. In Section 4.7 we held k_i constant but in this section we hold d_i constant. In both cases we have a penalty with a $\sqrt{\rho_i} = \sqrt{d_i/k_i}$ term.

However, we do not know the number of segments k_i in advance. But we supposed that the number of segments is proportional to the number of base pairs l_i, so we can use that in the penalty. This suggests a penalty that takes the form of $\sqrt{d_i/l_i}$. So in the next section, we confirm that this intuition works for constructing an optimal penalty.

4.10 Combining normalizations

In this section, we show that we can combine the results of the previous sections to create composite invariant penalties. In particular, to normalize for sampling density d_i and length in base pairs l_i, we need $\sqrt{d_i}$ and $1/\sqrt{l_i}$ terms in the penalty, respectively. This suggests that when considering variable d_i and l_i, we need a $\sqrt{d_i/l_i}$ term in the penalty, and in this section we show that this intuitive construction results in an optimal penalty.

In Figure 4.16, we plot 2 signals with different number of points d_i and length in base pairs l_i. We would like to find a penalty that allows us to generalize model complexity tradeoff parameters λ between these signals.

For each signal i, we define the penalty

$$k_i^\alpha(\lambda) = \arg\min_k \lambda k l_i^\alpha \sqrt{d_i} + ||y_i - \hat{y}_i^k||_2^2, \tag{4.40}$$

where l_i is the signal length in base pairs and d_i is the number of points sampled. We will attempt to determine an α that allows accurate breakpoint detection in signals of varying length and number of points sampled. Based on the result in Section 4.9, we expect to find $\alpha = -1/2$.

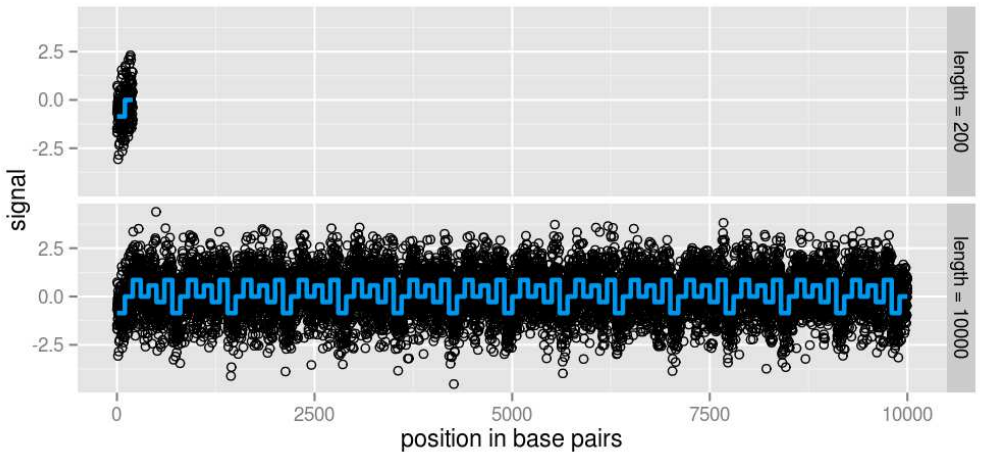

Figure 4.16: Two signals with a different number of points d_i and length in base pairs l_i.

In Figure 4.17, we plot the breakpoint error functions E_i^α for 2 signals i and several exponents α. The curves seem to align when $\alpha = -1/2$.

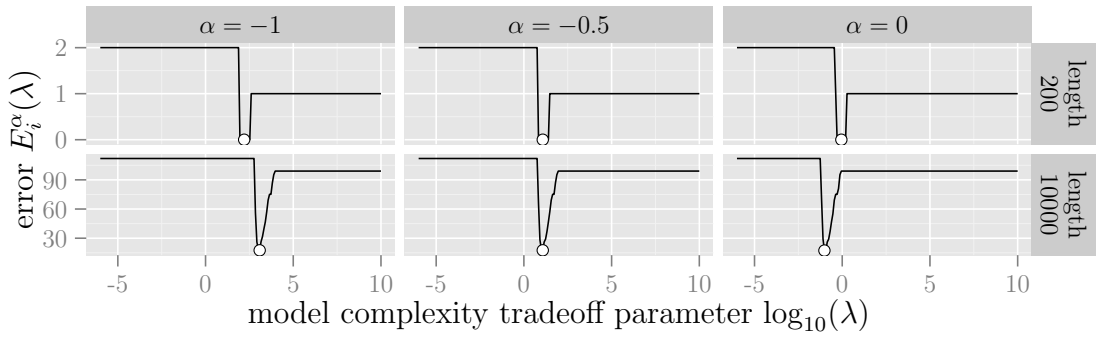

Figure 4.17: Breakpoint error functions E_i^α for several exponents α and 2 signals i of varying number of points d_i and length in base pairs l_i. The penalty contains a term $l_i^\alpha \sqrt{d_i}$.

In Figure 4.18, we plot the train and test error as a function of penalty exponent α. This analysis suggests that the optimal exponent is $\alpha = -1/2$, as expected from our previous analysis in Section 4.9.

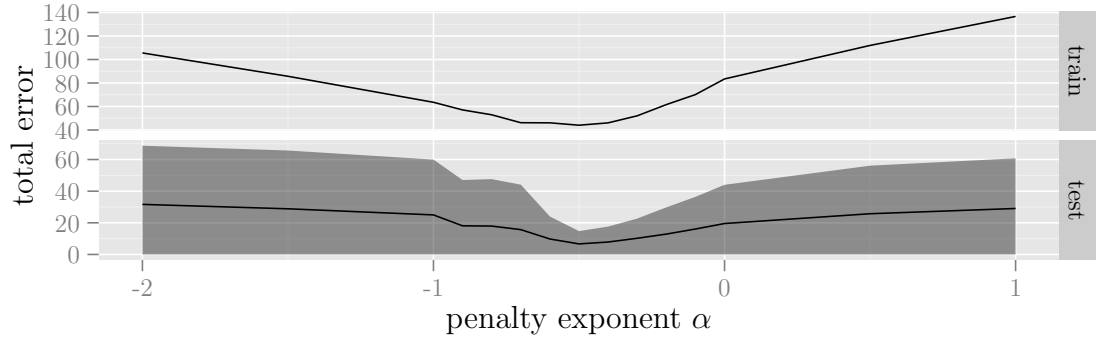

Figure 4.18: Train and test error functions for several signals of varying number of points d_i and length in base pairs l_i. The penalty contains a term $l_i^\alpha \sqrt{d_i}$.

4.11 Optimal penalties for the fused lasso signal approximator

In the previous sections, we used theoretical arguments and simulation experiments to determine the optimal penalties for cghseg. In this section, we demonstrate that the same approach can be used to find optimal penalties for another model, the Fused Lasso Signal Approximator (FLSA).

We used the `flsa` function in version 1.03 of the **flsa** package from CRAN to calculate the FLSA [Hoefling, 2009]. Let $x \in \mathbb{R}^d$ be the noisy copy number signal for one chromosome. The FLSA solves the following optimization problem:

$$\arg\min_{\mu \in \mathbb{R}^d} \frac{1}{2} \sum_{j=1}^{d} (x_j - \mu_j)^2 + \lambda_1 \sum_{j=1}^{d} |\mu_j| + \lambda_2 \sum_{j=1}^{d-1} |\mu_j - \mu_{j+1}|. \tag{4.41}$$

First, we take $\lambda_1 = 0$ since we are concerned with breakpoint detection, not signal sparsity. In this section, our aim is to determine a parameterization for λ_2 that we will be able to find similar breakpoints in signals of varying sampling density.

We use the same setup that we used to determine optimal penalties for cghseg, as described in Section 4.7 and shown again in Figure 4.19.

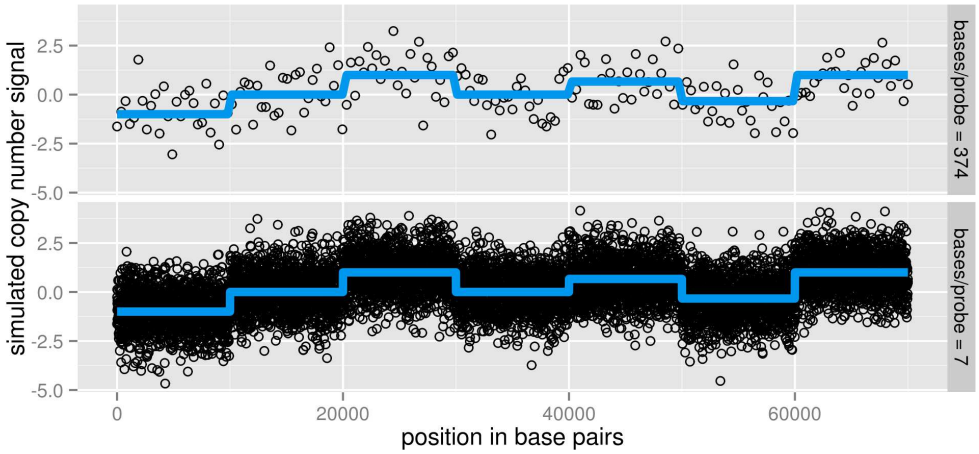

Figure 4.19: Simulated signals with different sampling density.

In particular, for every signal $i \in \{1, \ldots, n\}$, let $y_i \in \mathbb{R}^{d_i}$ be the noisy signals, sampled at positions $p_i \in \mathcal{X}^{d_i}$. To find an optimal penalty for these data, first let $\lambda_2 = \lambda d_i^\alpha$. For each signal i, exponent $\alpha \in \mathbb{R}$, and tradeoff parameter $\lambda \in \mathbb{R}^+$, we define the optimal smoothing as

$$\hat{y}_i^{\lambda,\alpha} = \underset{\mu \in \mathbb{R}^{d_i}}{\arg\min} \frac{1}{2}||y_i - \mu||_2^2 + \lambda d_i^\alpha \sum_{j=1}^{d_i - 1} |\mu_j - \mu_{j+1}|. \tag{4.42}$$

Then, we define the breakpoint detection error as a function of the breaks in the smoothed signal:

$$E_i^\alpha(\lambda) = E_{\text{exact}}^B \left[\varphi \left(\hat{y}_i^{\lambda,\alpha}, p_i \right) \right], \tag{4.43}$$

where the breakpoint function φ is defined in Equation 4.3 and the error E_{exact}^B is defined Equation 4.11.

We plot E_i^α for 2 signals i and several penalty exponents α in Figure 4.20. Note that the functions appear to align when $\alpha = 1$.

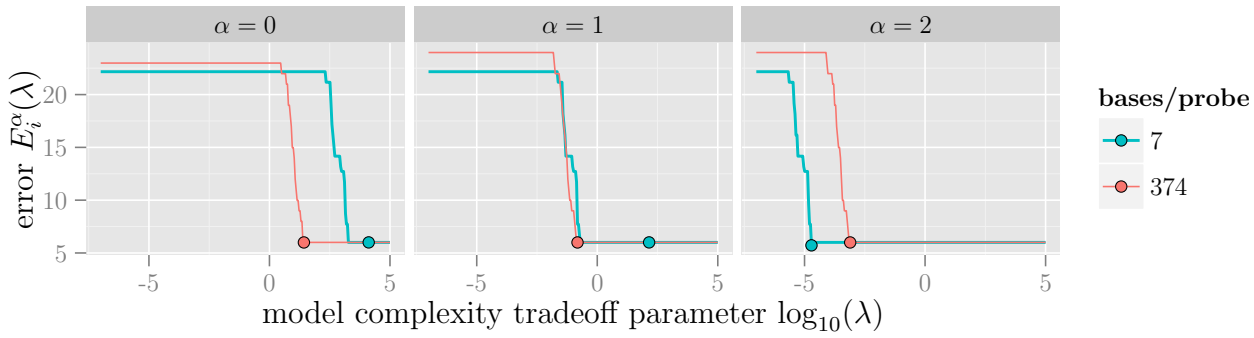

Figure 4.20: Model complexity breakpoint error functions E_i^α.

To evaluate which penalty parameter α results in optimal fitting and learning, we use E^* and TestErr as defined in Equations 4.30 and 4.32. These functions are plotted in Figure 4.21, and suggest that a value of $\alpha = 1$ is optimal. This analysis suggests that taking $\lambda_2 = \lambda d_i$ is optimal for breakpoint detection using FLSA. This agrees with the observation in Chapter 3 that the flsa.norm penalty with a d_i term works better than the un-normalized flsa penalty.

We conclude by noting that this procedure could also be applied to find penalties for FLSA that depend on estimated signal noise $\hat{s}_i^2$ and length in base pairs l_i. However, we did not pursue this since FLSA does not work as well as cghseg in practice on real data, as shown in Chapter 3.

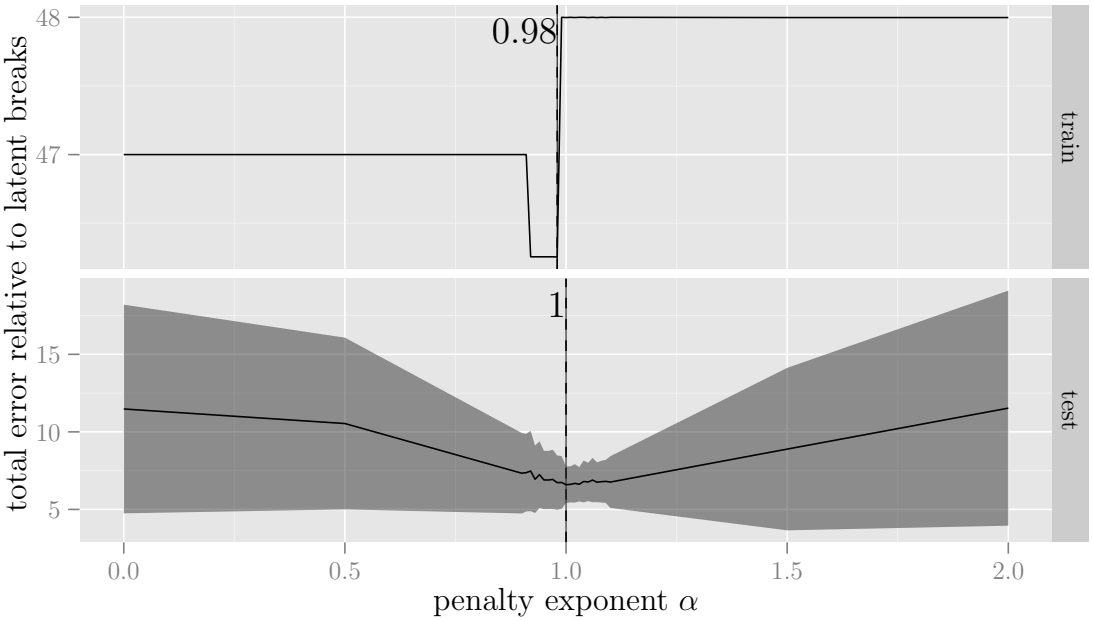

Figure 4.21: Train and test error as a function of penalty exponent α. The penalty has a term for the number of points sampled d_i^α.

4.12 Application to real data

In Sections 4.7-4.9, we found optimal cghseg penalties for data with varying sampling density, scale, and length. In Section 4.10, we also demonstrated that these results can be combined. This analysis suggests the following penalty, for every signal i:

$$k_i(\lambda) = \arg\min_k \lambda k \hat{s}_i^2 \sqrt{d_i/l_i} + ||y_i - \hat{y}_i^k||_2^2 \tag{4.44}$$

In Table 4.3, we report results of using the suggested penalties on the neuroblastoma data set. The normalizations suggested by the analysis of simulations do not improve breakpoint detection error in the neuroblastoma data set. This observation suggests that distribution that generates the real data is more complex than the simple simulation model considered in this chapter.

Practically speaking, we still would like to find a penalty with optimal breakpoint detection in real data. So, in the next chapter, we introduce a new method that uses interval regression to exploit the breakpoint annotations and to learn an optimal penalty from real data.

	points	length	variance	train	test.mean	test.sd
cghseg.k	1	0	0	2.19	2.20	1.01
cghseg.k.sqrt.d	1/2	0	0	3.51	3.87	1.58
cghseg.k.sqrt.d.var	1/2	0	2	3.19	4.47	5.02
cghseg.k.sqrt.var	1/2	−1/2	2	4.18	6.38	7.61

Table 4.3: Breakpoint detection error of cghseg on the neuroblastoma data set, with 1 row for each penalty. The exponent of the points d_i, length l_i, and variance $\hat{s}_i$ terms in the penalty is shown with the train and test error in percent.

Chapter 5

Learning a penalty for change-point detection using interval regression

The material from this chapter is taken from an article submitted for peer review. This is joint work with Guillem Rigaill with input from my advisors Francis Bach and Jean-Philippe Vert.

Chapter summary

In segmentation models, the number of segments is usually chosen using penalized cost functions that compromise between data fitting and model complexity. There are many penalties and many heuristics for choosing the constants in those penalties. In this work, we propose to learn the penalty and its constants in databases of signals with weak change-point annotations. We propose a convex relaxation that yields an interval regression problem, and solve it using accelated proximal gradient methods. Finally, we show that this method achieves state-of-the-art performance on a large database of annotated copy number profiles from neuroblastoma tumors.

5.1 Introduction

In the previous chapter, we found optimal penalties for learning model complexity parameters λ in simulated data with varying sampling density d_i, estimated noise σ_i, and length in base pairs l_i. That analysis suggested the penalty given in Equation 4.44, which did not work the best in real copy number profile data.

Presumably, the real data come from a probability distribution more complicated than the simple piecewise constant, homoscedastic simulations that we analyzed. So instead of using the penalty exponents α that we found using our analysis of simulations, in this chapter we develop a method to estimate these constants from real data.

Many penalties have been proposed for the change-point detection problem. The standard AIC or BIC criteria are not well adapted in this context since the model collection is exponential [Birgé and Massart, 2007, Schwarz, 1978, Akaike, 1973, Baraud et al., 2009], and also because change-points are discrete parameters [Zhang and Siegmund, 2007]. Many criteria specifically adapted to change-point models have been proposed. For example, there are many different variants of the BIC [Yao, 1988, Lee, 1995, Zhang and Siegmund, 2007], and the model selection theory of Birgé and Massart suggest other penalties [Birgé and Massart, 2007, Lavielle, 2005, Lebarbier, 2005]. These penalties are derived from theoretical considerations and give us insight into which features are important to select a good segmentation model. The exact shape or formula of these penalties depends on various assumptions such as normality and independence. These assumptions are often violated in real data, which can lead to selection of a suboptimal model. So in this chapter, rather than taking a particular penalty for granted, we propose to learn the penalty using annotation data.

In particular, we propose to learn a penalty from visual change-point annotations. By plotting the data, we can easily identify regions with changes and regions without changes. In Chapter 3, we proposed to use databases of visual annotations to calibrate standard model selection criteria up to a constant λ determined by grid search. This chapter generalizes that approach by learning a penalty as a function of features of the signal.

Since the penalty learning problem involves an intractable optimization, we propose a convex relaxation. We propose to solve the resulting interval regression problem using accelated proximal gradient methods, which permit efficient inference of the support and constants in the penalty function.

The structure of this chapter is as follows. In Section 5.2, we describe the penalty learning problem using the non-convex annotation error. Then, we propose a convex relaxation in Section 5.3, and describe the necessary optimization algorithms in Section 5.4. Finally, in Section 5.5 we show results on a large database of neuroblastoma copy number profiles.

5.2 The penalty learning problem

Assume we have a set of n annotated training signals. Two signals from the neuroblastoma data set are shown in the top panel of Figure 5.1. For every training signal $i \in \{1, \dots, n\}$, let $y_i \in \mathbb{R}^{d_i}$ be the noisy signal sampled at positions p_i, sorted such that $p_{i1} < \cdots < p_{i,d_i}$.

We use the pruned dynamic programming algorithm to calculate the maximum likelihood segmentations $\hat{y}_i^k \in \mathbb{R}^{d_i}$ for each model size $k \in \{1, \dots, k_{\max}\}$ [Rigaill, 2010]. The change-point indices are

$$J_i^k = \{j \in \{1, \dots, d_i - 1\} \mid \hat{y}_{ij} \neq \hat{y}_{i,j+1}\} \tag{5.1}$$

and we estimate the positions after which a change-point occured using the mean

$$\hat{P}_i^k = \{\lfloor (p_{ij} + p_{i,j+1})/2 \rfloor \forall j \in J_i^k\}. \tag{5.2}$$

Change-point annotations define a non-convex error function

Let R_i, A_i be the sets of regions and annotations used to calculate the change-point detection error for signal i. Every annotation $a \in A_i$ is a set that specifies the number of changes in the corresponding region $r \in R_i$. The annotation error $e_i : \{1, \dots, k_{\max}\} \to \mathbb{R}^+$ compares the estimated number of changes in each region $|\hat{P}_i^k \cap r|$ to the annotated number of changes a using the zero-one loss:

$$e_i(k) = \sum_{(r,a)\in(R_i,A_i)} 1_{|\hat{P}_i^k \cap r| \notin a}. \tag{5.3}$$

For every signal i, we define the optimal number of segments as

$$k_i^*(g) = \underset{k\in\{1,\dots,k_{\max}\}}{\arg\min}\ g(k, x_i) + ||y_i - \hat{y}_i^k||_2^2, \tag{5.4}$$

where the penalty g is a function of the number of segments k and some features $x_i \in \mathbb{R}^m$ that do not depend on the annotation.

The problem we tackle in this chapter is to use the n annotated signals to learn the best penalty g for change-point detection:

$$\min_g \sum_{i=1}^{n} e_i \left[k_i^*(g)\right], \tag{5.5}$$

For all that follows we will consider linear penalty functions g of the form $g^h(k, x_i) = h(x_i)k$, yielding the problem

$$\min_h \sum_{i=1}^{n} e_i \left[k_i^* \left(g^h\right)\right]. \tag{5.6}$$

This simplification excludes penalty functions with nonlinear k terms [Lebarbier, 2005]. However, this allows efficient learning by first calculating

$$z_i^*(L) = \underset{k\in\{1,\dots,k_{\max}\}}{\arg\min}\ \exp(L)k + ||y_i - \hat{y}_i^k||_2^2. \tag{5.7}$$

This is a function $z_i^* : \mathbb{R} \to \{1, \dots, k_{\max}\}$ that uses the linear penalty with tradeoff $\exp(L)$ to select the number of segments for signal i. In the bottom of Figure 5.1, we show two functions z_i^*, and their corresponding annotation error functions $E_i : \mathbb{R} \to \mathbb{R}^+$, defined as

$$E_i(L) = e_i \left[z_i^*(L)\right]. \tag{5.8}$$

Note that z_i^* and E_i are non-convex, piecewise constant functions that can be efficiently calculated prior to learning, using the algorithm we discuss in Section 5.4.

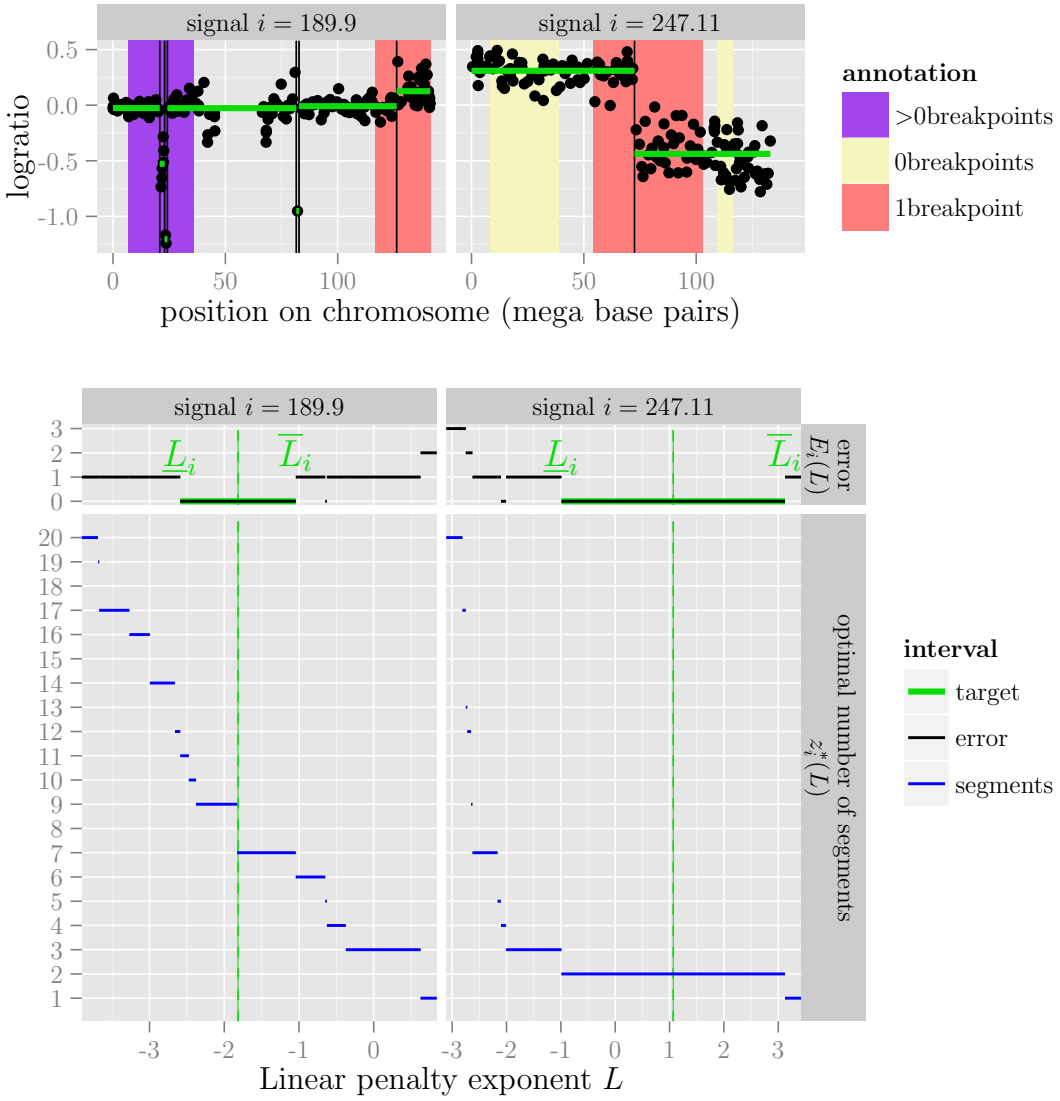

Figure 5.1: **Top**: two signals y_i (black points) with their annotations (rectangles). A consistent model $\hat{y}_i^k$ (green lines) is shown with its breakpoints $\hat{P}_i^k$ (vertical black lines). **Bottom**: the optimal number of segments z_i^* and annotation error curves E_i. The target interval $[\underline{L}_i, \overline{L}_i]$ is drawn in green, with a vertical dashed line to indicate the complexity of the model $\hat{y}_i^k$ plotted above.

By definition, we have the following relation between the annotation error functions:

$$\begin{aligned} e_i\left[k_i^*\left(g^h\right)\right] &= e_i\left[z_i^*\big(\log h(x_i)\big)\right] && (5.9)\\ &= E_i\left[\log h(x_i)\right]. && (5.10)\end{aligned}$$

Rewriting the learning problem using E_i, we obtain

$$\min_h \sum_{i=1}^{n} E_i\left[\log h(x_i)\right]. \tag{5.11}$$

Log-linear penalty functions

Now, we need to specify what kind of penalty function h we will learn. Although non-parametric models such as k-nearest neighbors could be used, there are several interesting penalties defined by supposing that h is log-linear: $\log h(x_i) = w'x_i + \beta$. This results in the model selection criterion

$$z_i^*(w'x_i + \beta) = \arg\min_k ||y_i - \hat{y}_i^k||_2^2 + k\exp(\beta + w'x_i), \tag{5.12}$$

and we compare several special cases of Equation 5.12 in Table 5.1.

Model	Penalty $g(k, x_i)$	Learned Penalty $h(x_i)$	Parameters	Features x_i
Lavielle	$\alpha k d_i$	$\alpha d_i^{w_1}$	$\alpha \in \mathbb{R}^+$	
			$w_1 \in \mathbb{R}$	$\log d_i$
Chapter 4	$\alpha k \sigma_i^2 \sqrt{d_i/l_i}$	$\alpha \sigma_i^{w_1} d_i^{w_2} l_i^{w_3}$	$\alpha \in \mathbb{R}^+$	
			$w_1 \in \mathbb{R}$	$\log \sigma_i$
			$w_2 \in \mathbb{R}$	$\log d_i$
			$w_3 \in \mathbb{R}$	$\log l_i$
BIC	$2k\sigma_i^2 \log d_i$	$\alpha \sigma_i^{w_1} (\log d_i)^{w_2}$	$\alpha \in \mathbb{R}^+$	
			$w_1 \in \mathbb{R}$	$\log \sigma_i$
			$w_2 \in \mathbb{R}$	$\log \log d_i$
Lebarbier	$\alpha k \sigma_i^2 (c_1 \log(d_i/k) + c_2)$	$\approx \alpha \sigma_i^{w_1} (2 \log d_i + 5)^{w_2}$	$\alpha \in \mathbb{R}^+$	
			$w_1 \in \mathbb{R}$	$\log \sigma_i$
			$w_2 \in \mathbb{R}$	$\log(2 \log d_i + 5)$
General		$\exp(w'x_i + \beta)$	$\beta = \log \alpha \in \mathbb{R}$	
			$w \in \mathbb{R}^m$	$x_i \in \mathbb{R}^m$

Table 5.1: Some penalties that we can learn using log-linear penalty functions $\log h(x_i) = w'x_i + \beta$.

For example, Lavielle [2005] suggested a penalty of the form $d_i k$, where d_i is the number of points sampled from signal i. This is equivalent to using just one feature $x_i = \log d_i$ and taking $h(x_i) = \alpha d_i^{w_1}$, where $\alpha \in \mathbb{R}$ and $w_1 \in \mathbb{R}$ are penalty parameters to learn. In the cghseg.k model that showed the best breakpoint detection in the neuroblastoma data, the authors fixed $w_1 = 1$ [Hocking et al., 2012]. In contrast, in this formulation we consider it as an optimization variable.

The simulations in Chapter 4 suggest a penalty with terms for the length of the signal in base pairs l_i and the estimated variance of the signal σ_i. The penalty found to be optimal in simulations of signals with homoscedastic Gaussian noise was $\alpha k \sigma_i^2 \sqrt{d_i / l_i}$, where the constant α is learned by minimizing the breakpoint error. We can instead learn the penalty function in annotated signals by taking three features $x_i = \begin{bmatrix} \log \sigma_i & \log d_i & \log l_i \end{bmatrix}$, which corresponds to learning a penalty $h(x_i) = \alpha \sigma_i^{w_1} d_i^{w_2} l_i^{w_3}$.

As another example, the well-known Bayesian Information Criterion due to Schwarz [1978] uses $2k\sigma_i^2 \log d_i$ as a penalty. This corresponds to choosing two features $x_i = \begin{bmatrix} \log \sigma_i & \log \log d_i \end{bmatrix}'$ and taking $h(x_i) = \alpha \sigma_i^{w_1} (\log d_i)^{w_2}$.

According to Lebarbier [2005], the optimal penalty has constants c_1 and c_2. But since this penalty is not linear in the number of segments k, log-linear models can not be applied to estimate these constants. Instead, we can use the constants suggested by their simulations $c_1 = 2$ and $c_2 = 5$ to construct a feature $x_i = \log(2 \log d_i + 5)$, and use that in a log-linear model.

However, in any of these models, the learning is still intractable. Since the annotation error E_i is a non-convex piecewise constant function, the minimization in problem 5.11 can only be accomplished via exhaustive search. For one or two variables this may be feasible using grid search. However, for multivariate models, grid search is very inefficient. So instead of minimizing the annotation error E_i directly, we propose a convex relaxation in the next section that yields an efficient interval regression algorithm for finding the intercept $\beta \in \mathbb{R}$ and weights $w \in \mathbb{R}^m$.

5.3 A convex relaxation of the annotation error

In this section, we develop a surrogate loss l_i that is a convex relaxation of the annotation error E_i. In particular, we propose to make learning problem (5.11) tractable using these two modifications:

- Instead of minimizing $E_i(L)$ directly, we define a target interval of L values, yielding an interval regression problem.
- We replace the non-convex annotation error E_i with a margin-based convex surrogate loss l_i.

The interval regression problem

We begin by defining a target interval $[\underline{L}_i, \overline{L}_i]$ of $E_i(L)$, for every signal i. We take the notation conventions from the interval analysis literature [Nakao et al., 2010]. Note in Figure 5.1 that there may be more than one interval that achieves the minimum of E_i, so we define the target interval as the largest continuous minimum. Note also that depending on the annotation data for signal i, it is possible to have $\underline{L}_i = -\infty$ or $\overline{L}_i = \infty$, as shown in the top panel of Figure 5.2.

Restating the learning problem in terms of the target interval, the goal is to find a regression function $f : \mathbb{R}^m \to \mathbb{R}$ such that $\underline{L}_i < f(x_i) < \overline{L}_i$ for every signal i. In the middle panel of Figure 5.2 we plot the target intervals $[\underline{L}_i, \overline{L}_i]$ as a function of one feature, a variance estimate $\log \sigma_i$. Geometrically, the learning problem corresponds to finding a function f that intersects each of the target intervals.

Another interpretation is shown in the bottom panel of Figure 5.2, where we plot just the limits $\underline{L}_i, \overline{L}_i$ of the target interval. Here, the learning problem corresponds to finding a function that separates the two classes of points. However, this is not the same problem as the Support Vector Machine, as will be explained in the next section.

It is important to note that when the number of data points n is small relative to the number of features m, there can be infinitely many lines that achieve zero breakpoint detection error. However, for the purposes of learning it will be best to choose a separator with a large margin, as shown with the dashed red horizontal line in the bottom of Figure 5.2.

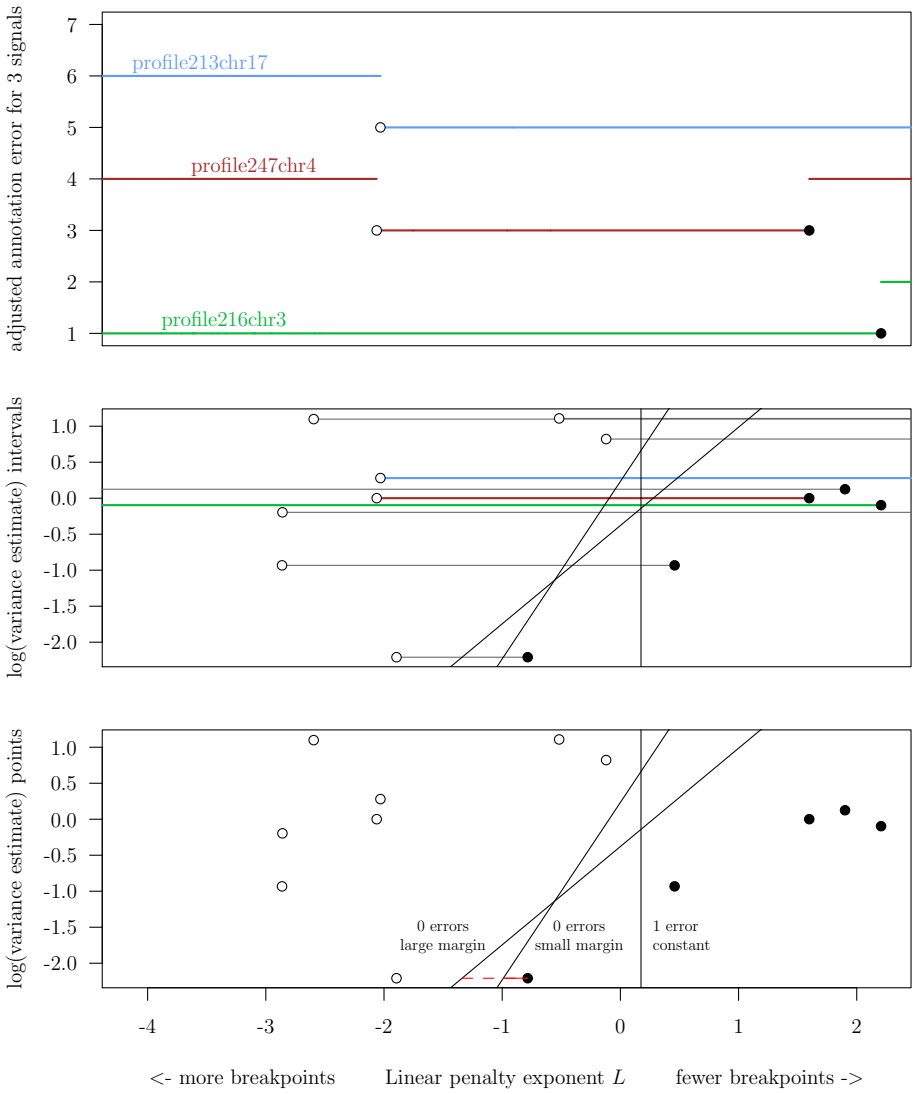

Figure 5.2: Idea of interval regression for predicting model complexity parameters. **Top**: annotation error curves E_i for 3 signals i. **Middle**: for several more signals i, we show only the interval of minimal error $[\underline{L}_i, \overline{L}_i]$, plotted using a variance estimate σ_i on the vertical axis. A regression line achieves minimal annotation error if it intersects all the intervals. **Bottom**: only the limits $\underline{L}_i$ and $\overline{L}_i$ of each interval are plotted. A regression line achieves 0 annotation error if it separates the $\underline{L}_i$ from the $\overline{L}_i$. The margin is drawn in red for two separating regression lines.

Maximum margin regression line for separable data

For each training signal i, we have a feature vector $x_i \in \mathbb{R}^m$. We get the best segmentation on the training set if we can find a function $f : \mathbb{R}^m \to \mathbb{R}$ such that $\underline{L}_i < f(x_i) < \overline{L}_i$ for all training signals i.

If the data are linearly separable using the features x_i, we can find infinitely many affine functions f that have minimal annotation error. However, for learning it will be best to choose the separator of maximal margin, which we construct by solving the problem

$$\begin{aligned} \underset{\beta\in\mathbb{R},w\in\mathbb{R}^m,\mu\in\mathbb{R}^+}{\text{maximize}} \quad & \mu \\ \text{subject to} \quad & w'x_i + \beta - \underline{L}_i \geq \mu, \text{ for all } i \text{ such that } \underline{L}_i > -\infty \\ & \overline{L}_i - w'x_i - \beta \geq \mu, \text{ for all } i \text{ such that } \overline{L}_i < \infty. \end{aligned} \tag{5.13}$$

Note this is a Linear Program (LP) so the solution can be found using any generic LP solver. An LP is a special case of a Quadratic Program (QP), so a QP solver can also be used. I used the dual method of Goldfarb and Idnani [1983], which is implemented in the `solve.QP()` function in the **quadprog** R package [Turlach and Weingessel, 2011]. To use the solver, we need to convert the problem to standard form:

$$\begin{aligned} \underset{\beta\in\mathbb{R},w\in\mathbb{R}^m,\mu\in\mathbb{R}}{\text{minimize}} \quad & -\mu \\ \text{subject to} \quad & -\mu + w'x_i + \beta \geq \underline{L}_i, \text{ for all } i \text{ such that } \underline{L}_i > -\infty \\ & -\mu - w'x_i - \beta \geq -\overline{L}_i, \text{ for all } i \text{ such that } \overline{L}_i < \infty \\ & \mu \geq 0. \end{aligned} \tag{5.14}$$

The regression function found by solving problem 5.14 for a small separable data set with 1 variance estimate feature $x_i = \log \sigma_i \in \mathbb{R}$ is shown in Figure 5.3. It is important to note that the geometric interpretation of the margin is not the same as the usual Support Vector Machine for binary classification. In fact, the margin is the distance along the L axis between the regression line and the closest limits $\underline{L}_i, \overline{L}_i$.

However, any sizable real data set will not be separable. So in the next section, we use max-margin idea to develop a surrogate loss for interval regression on real data sets.

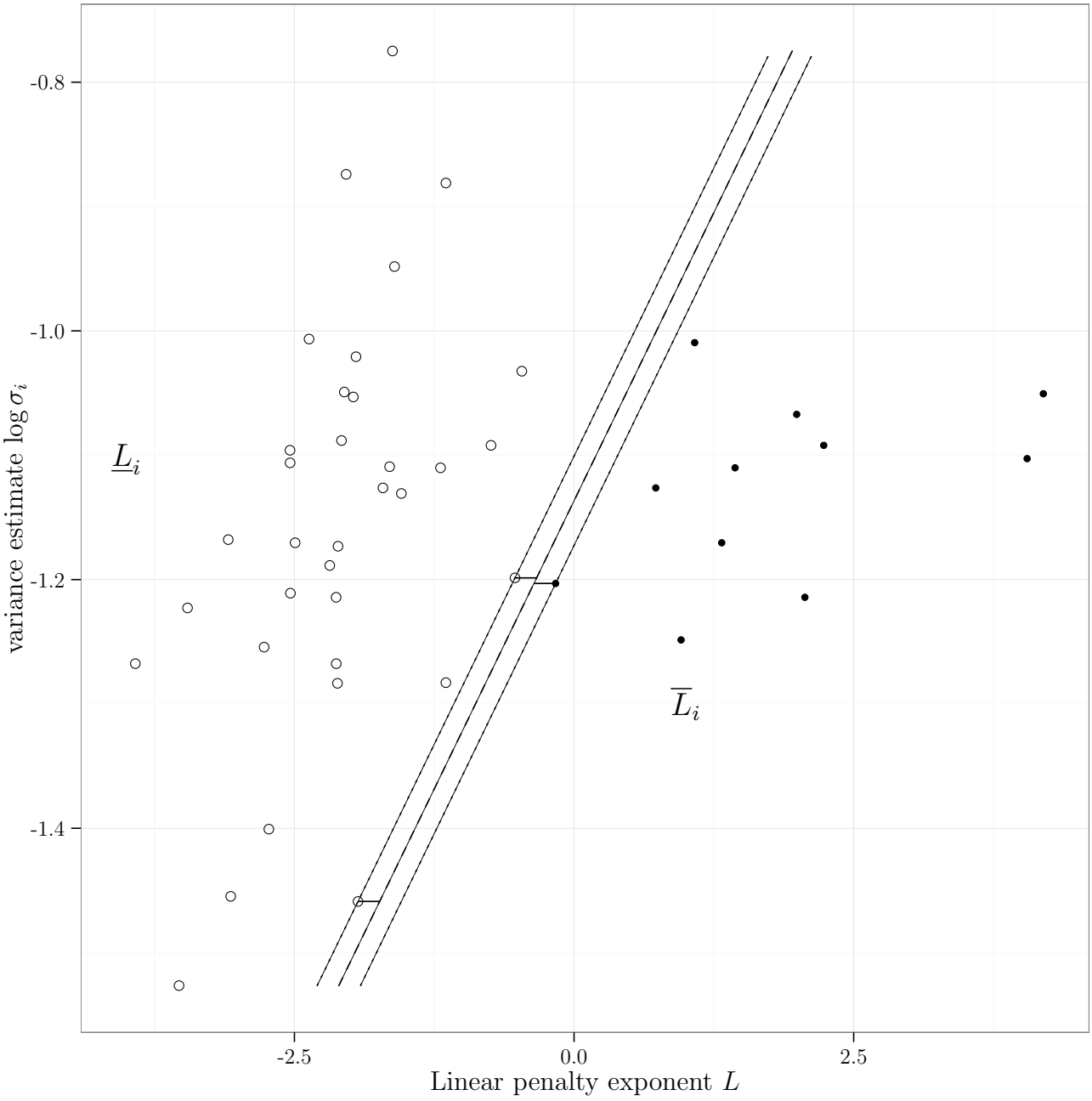

Figure 5.3: The maximum margin interval regression line is found by solving problem (5.13), and is drawn as a dashed line. The limits $\underline{L}_i, \overline{L}_i$ of target intervals are drawn using points for a small data set that is linearly separable using the variance estimate feature $x_i = \log \sigma_i$. The horizontal margin μ is drawn for the 3 border points.

Surrogate loss for non-separable data

Let us consider the class of surrogate loss functions $l_i : \mathbb{R} \to \mathbb{R}^+$ defined by

$$l_i(L) = \varphi\left(\frac{L - \underline{L}_i}{\delta}\right) + \varphi\left(\frac{\overline{L}_i - L}{\delta}\right), \tag{5.15}$$

where the binary classification surrogate loss function $\varphi : \mathbb{R} \to \mathbb{R}^+$ is a convex relaxation of the zero-one loss. The parameter $\delta > 0$ controls the size of the margin, and we used $\delta = 1$ since that worked well in the data we analyzed. Using the hinge loss for φ results in a surrogate loss similar to the ϵ-insensitive loss used for Support Vector Regression [Vapnik et al., 1997]. Some other choices for φ include log and Huber losses, but we used the squared hinge loss since it exhibited the best learning:

$$\varphi(L) = \begin{cases} (L-1)^2 & \text{if } L \leq 1 \\ 0 & \text{if } L \geq 1. \end{cases} \tag{5.16}$$

Note that l_i is convex since it is the sum of two convex functions. This convex relaxation can clearly be seen in Figure 5.4, where we plot the surrogate loss l_i along with the annotation error E_i for several signals i. Let the average surrogate loss be

$$\text{loss}(\beta, w) = \frac{1}{n}\sum_{i=1}^{n} l_i(w' x_i + \beta). \tag{5.17}$$

To encourage a sparse weight vector w, we use an ℓ_1 penalty with the surrogate loss, which yields the optimization problem

$$\underset{\beta\in\mathbb{R}, w\in\mathbb{R}^m}{\text{minimize}}\, \gamma||w||_1 + \text{loss}(\beta, w), \tag{5.18}$$

where $\gamma \in \mathbb{R}^+$ is a fixed value that controls the degree of regularization. Note that the ℓ_1 norm encourages some entries of w to be exactly zero, which has the effect of selecting which features are used in the penalty function $h(x_i) = \exp(w' x_i + \beta)$.

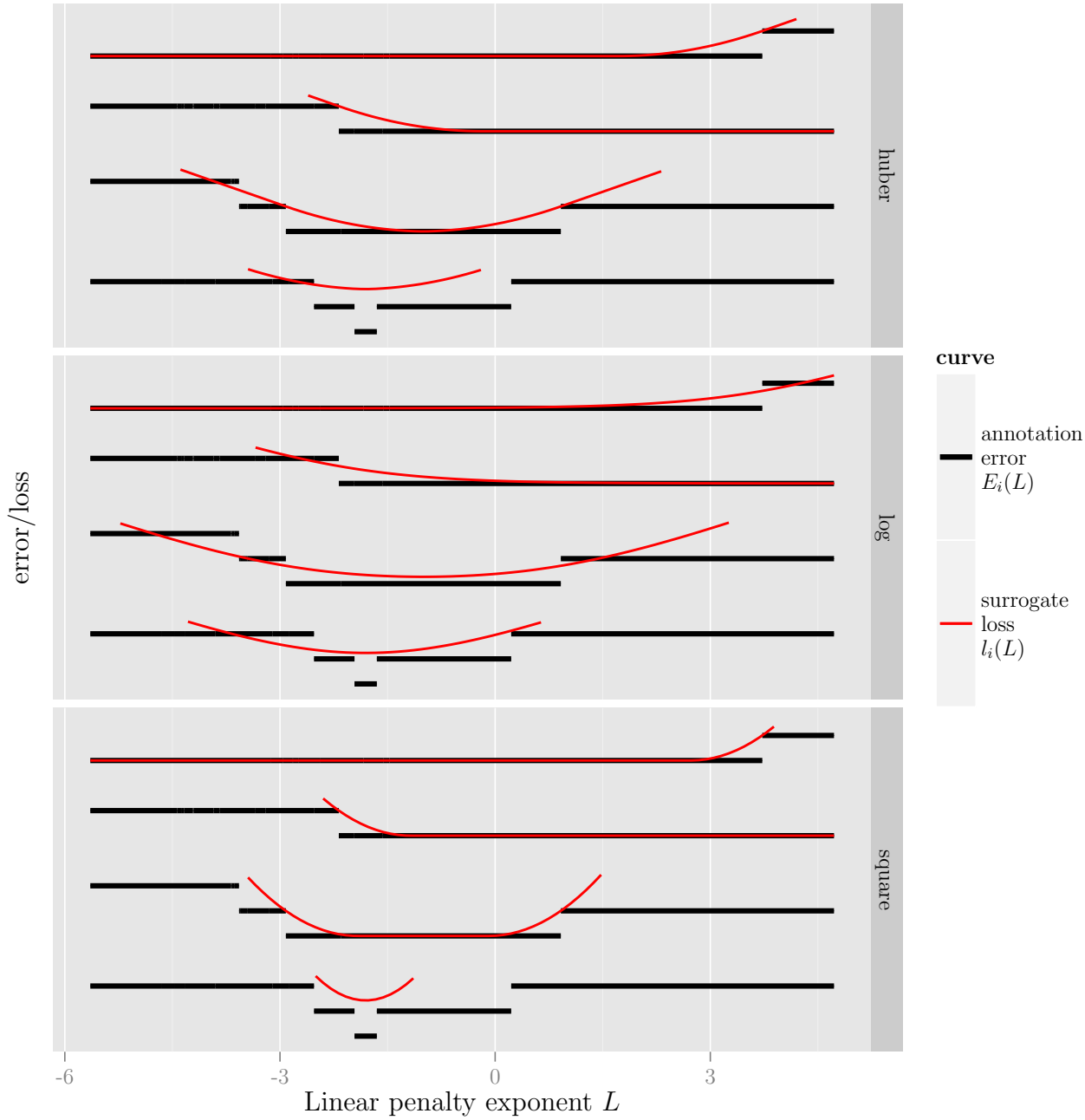

Figure 5.4: Several relaxations of the annotation error (panels from top to bottom). Each panel shows the annotation error E_i and its convex relaxation l_i for four signals i.

5.4 Algorithms

In this section, we will describe the algorithms that implement the penalty function learning methods that we have described in the previous sections. First, we recover an exact representation of the model selection functions z_i^*, which we use to calculate the annotation error functions E_i and the target intervals $[\underline{L}_i, \overline{L}_i]$. Then, we show how accelerated proximal gradient methods may be used to learn a penalty function from these target intervals.

Calculating the exact model selection functions

For a given profile i and number of segments k, we define the model selection criterion

$$\text{crit}_i^k(\lambda) = \lambda k + ||y_i - \hat{y}_i^k||_2^2 \tag{5.19}$$

and we note that each crit_i^k is an affine function of λ. In Figure 5.5, we plot each function crit_i^k as a line for every model size $k \in \{1, \ldots, 20 = k_{\max}\}$ for one signal i.

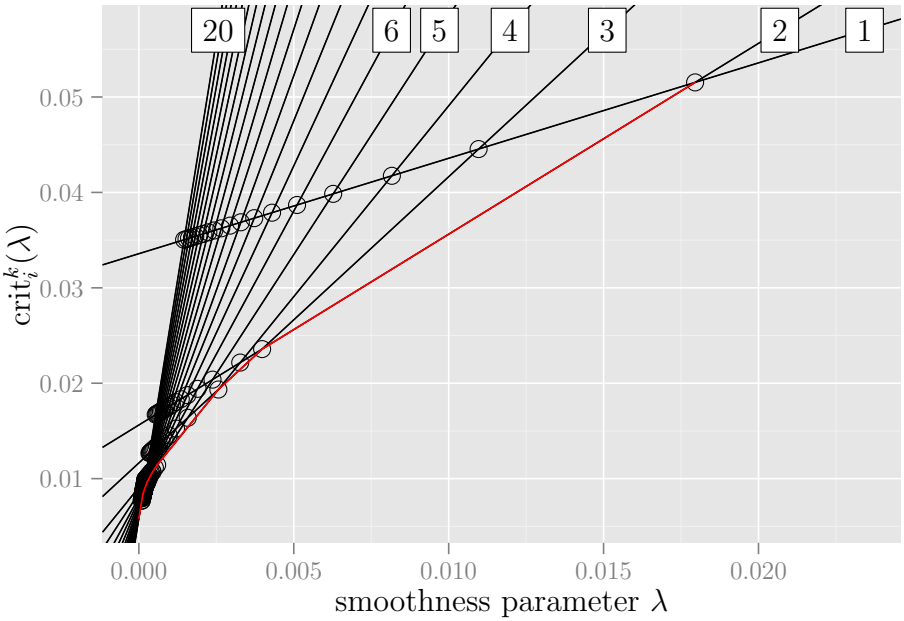

Figure 5.5: Calculating the exact path of optimal number of segments z_i^* (red lines). The functions crit_i^k are drawn as lines, and $k \in \{1, \ldots, 6, 20\}$ is shown. The intersections are drawn as points.

Then we note that

$$z_i^*(\log \lambda) = \underset{k \in \{1,\dots,k_{\max}\}}{\arg\min} \quad \text{crit}_i^k(\lambda), \tag{5.20}$$

so to find the exact path of solutions z_i^* we simply need to find the minimum of a finite set of affine functions.

The algorithm initializes the current number of segments k_c to the largest plausible k, since $||y_i - \hat{y}_i^k||_2^2$ is a decreasing function of k. Then the algorithm computes for all larger plausible k their hit-times with k_c, meaning the smallest λ for which k is preferred over k_c. Then the algorithm stores the smallest of these λ and the corresponding number of changes (nextλ, nextK). The algorithm updates k_c to next$K$ and continues until $k_c = 1$.

The complexity of this algorithm is $O(k_{\max}^2)$. In practice, calculating the model selection functions z_i^*, E_i using Algorithm 3 is much faster than calculating the segmentation $\hat{y}_i^k$ using pruned DP.

Algorithm 3 EXACT-BREAKS

Input: model squared error $||y_i - \hat{y}_i^k||_2^2$ for all $k \in \{1, \dots, k_{\max}\}$.
$k_c \leftarrow \max\{\text{plausibleK}\}$
plausibleK $\leftarrow$ plausibleK $\setminus\, k_c$
while plausibleK $\neq \emptyset$ **do**
  next$\lambda \leftarrow +\infty$, nextK $\leftarrow 0$
  for $k \in$ plausibleK **do**
    $$\text{hit\_time} \leftarrow \frac{||y_i - \hat{y}_i^{k_c}||_2^2 - ||y_i - \hat{y}_i^k||_2^2}{k - k_c}$$
    if next$\lambda >$ hit_time **then**
      next$\lambda \leftarrow$ hit_time, nextK $\leftarrow k$
    end if
  end for
  $k_c \leftarrow$ nextK, **Save** k_c, nextλ
  plausibleK $\leftarrow$ plausibleK $\setminus \{k \mid k \geq k_c\}$
end while
Output: Optimal number of segments function z_i^*, represented by the set of saved k_c, nextλ.

We can fix a value of L and evaluate $z_i^*(L)$ to verify that the exact path calculation works correctly. We show these two calculations in Figure 5.6, which indeed shows that the exact path algorithm is correct.

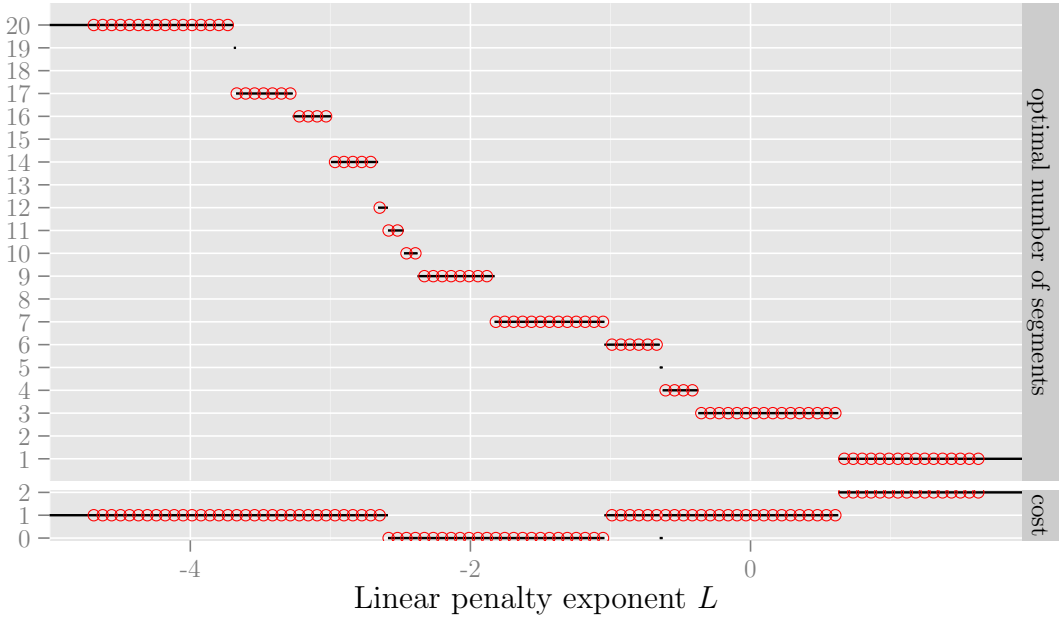

Figure 5.6: Black lines show the exact functions z_i^* calculated using the Exact-breaks algorithm, and red points show the approximation found by evaluating $z_i^*(L)$ on a grid of L values.

Optimization via proximal gradient methods

To efficiently solve problem 5.18, we can use FISTA [Beck and Teboulle, 2009]. To use FISTA to solve this problem, we need the partial derivatives of the surrogate loss:

$$\frac{\partial}{\partial \beta} l_i(w'x_i + \beta) = \varphi'(w'x_i + \beta - \underline{L}_i) - \varphi'(\overline{L}_i - w'x_i - \beta) \tag{5.21}$$

$$\frac{\partial}{\partial w_j} l_i(w'x_i + \beta) = x_{ij}\left(\varphi'(w'x_i + \beta - \underline{L}_i) - \varphi'(\overline{L}_i - w'x_i - \beta)\right). \tag{5.22}$$

The squared hinge loss φ has the following derivative:

$$\varphi'(L) = \begin{cases} 2(L-1) & \text{if } L \leq 1 \\ 0 & \text{if } L \geq 1. \end{cases} \tag{5.23}$$

We can use an approximate subdifferential optimality condition for a stopping criterion. The exact subdifferential optimality condition is

$$\nabla \text{loss}(\beta, w) \in \gamma \partial ||w||_1, \tag{5.24}$$

but on computers we will never exactly achieve this condition. So the approximate optimality condition that we use in practice is

$$\left| \frac{\partial}{\partial \beta} \text{loss}(\beta, w) \right| \leq \epsilon, \tag{5.25}$$

and for every variable $j \in \{1, \ldots, m\}$,

$$\begin{cases} \left| -\gamma + \frac{\partial}{\partial w_j} \text{loss}(\beta, w) \right| \leq \epsilon & \text{if } w_j < 0 \\ \left(\left| \frac{\partial}{\partial w_j} \text{loss}(\beta, w) \right| - \gamma \right)_+ \leq \epsilon & \text{if } w_j = 0 \\ \left| \gamma + \frac{\partial}{\partial w_j} \text{loss}(\beta, w) \right| \leq \epsilon & \text{if } w_j > 0 \end{cases} \tag{5.26}$$

for some positive constant $\epsilon > 0$ that controls how far we are from an optimal solution. For learning it is not necessary to use a very small threshold [Bach et al., 2012], so in our experiements we used $\epsilon = 10^{-3}$, which was sufficient.

Note that to apply the FISTA method we need to solve the proximal operator $p_\eta : \mathbb{R}^{p+1} \to \mathbb{R}^{p+1}$, which in our case is

$$p_\eta(\beta, w) = \begin{bmatrix} \beta - \frac{1}{\eta} \frac{\partial}{\partial \beta} \text{loss}(\beta, w) \\ s_{\gamma/\eta} \left(w_1 - \frac{1}{\eta} \frac{\partial}{\partial w_1} \text{loss}(\beta, w) \right) \\ \vdots \end{bmatrix} \tag{5.27}$$

where η is a Lipshitz constant of the smooth loss. We use a constant $\eta = m + \sqrt{m}$ which is heuristic but worked well on the data we analyzed. The soft-thresholding function $s_\lambda : \mathbb{R} \to \mathbb{R}$ is defined as

$$s_\lambda(x) = \begin{cases} 0 & \text{if } |x| < \lambda \\ x - \lambda \, \text{sign}(x) & \text{otherwise.} \end{cases} \tag{5.28}$$

To train the ℓ_1 regularized model, we need to select a degree of regularization γ. To do this, we split the data into a train and test set, and on the train set we fit a sequence of ℓ_1 regularized interval regression models. We plot the annotation error and surrogate loss of the trained model as a function of regularization in Figure 5.7. To select γ, we minimize the test annotation error. In the example shown, this results in selecting a coefficient for the log.bases variable which is exactly zero. This training protocol yields a sparse model that selects a subset of input features to use in the learned penalty function.

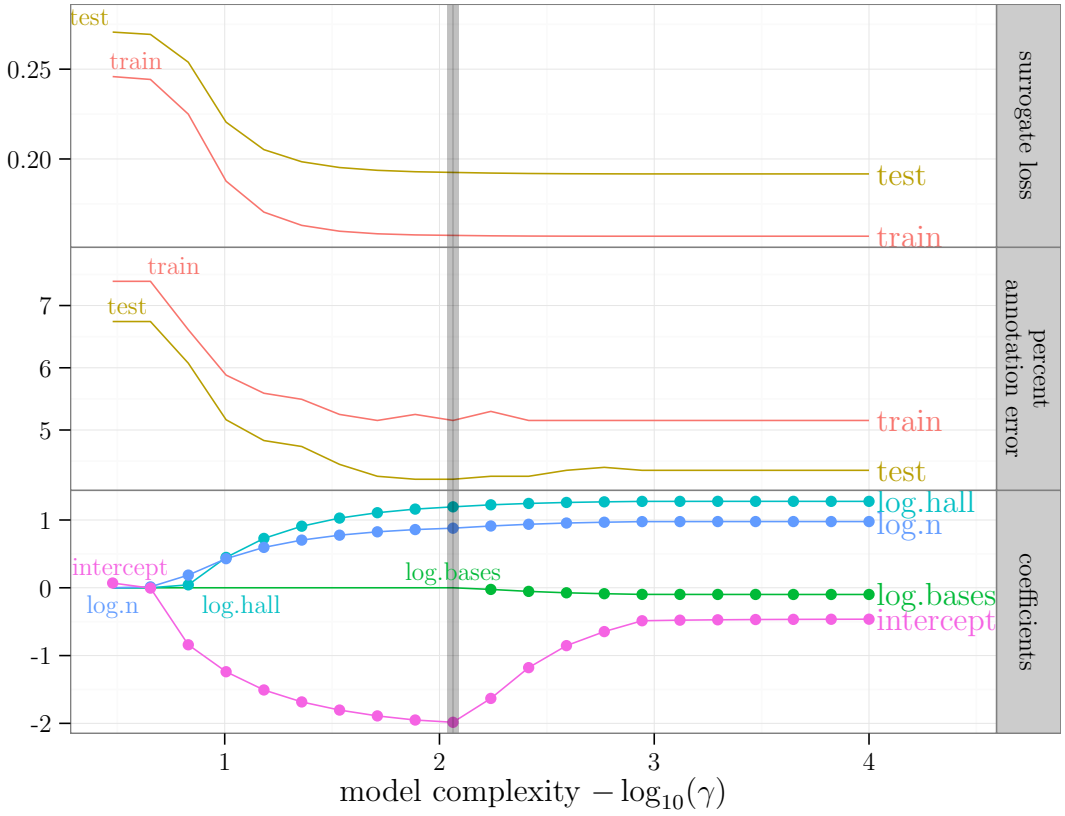

Figure 5.7: **Top**: surrogate loss(β^γ, w^γ) of the model with regularization γ, averaged over signals i in the training and test sets. **Middle**: train and test annotation error $\sum_i E_i(x_i' w^\gamma + \beta^\gamma)$, with a shaded grey vertical bar that indicates the minimum test error. **Bottom**: estimated coefficients β^γ, w_j^γ, with dots indicating non-zero values.

5.5 Results and discussion

We used Algorithm 3 to calculate the target intervals for 3 annotation data sets based on the neuroblastoma data. In Figure 5.8, the scatterplots of target intervals show a clear dependence on the estimated noise σ_i, which is not modeled using the state-of-the-art cghseg.k model.

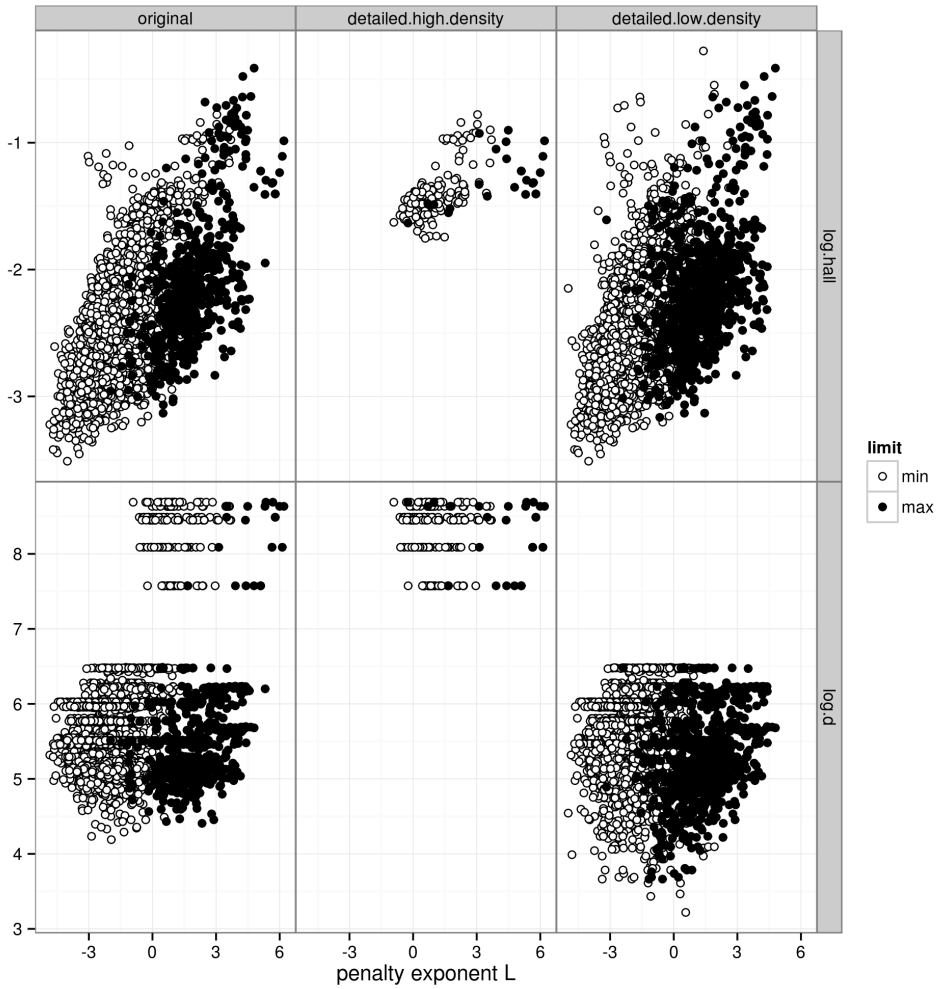

Figure 5.8: For several annotations of the neuroblastoma data (panels from left to right), we plot the limits of the target interval as a function of a variance estimate (log.hall) or signal dimension (log.d).

Learned penalty functions

We learned penalty functions on these three data sets using four different models. We focus on models of the form:

$$f(x_i) = w_1 \log \sigma_i + w_2 \log d_i + \beta, \tag{5.29}$$

where the features x_i include the number of points sampled d_i and a difference-based variance estimate σ_i [Hall et al., 1990]. We compared 3 un-regularized versions of this model, and one multivariate model with 117 features selected using ℓ_1-regularization:

- **cghseg.k** takes $w_1 = 0$ and $w_2 = 1$, then learns β by minimizing E_i using grid search.
- **log.d** takes $w_1 = 0$, and learns w_2 and β by minimizing the un-regularized surrogate loss.
- **log.s.log.d** learns w_1, w_2, β by minimizing the un-regularized surrogate loss. We report the coefficients learned in this model in Table 5.2, and it is interesting to note that the coefficients are clearly not the same across data sets. Note that in the original data set the optimal penalty was found to have a $d_i^{0.96}$ term, which explains why cghseg.k worked well in these data. Also, the models do not show the σ_i^2 term that is suggested by model selection theory.
- **L1-reg** constructs a feature vector $x_i \in \mathbb{R}^{117}$ consisting of features such as variance estimates, signal size measurements (d_i, $\log d_i$, ...), model RSS and MSE, and indicator variables for each chromosome. Then we use an internal cross-validation loop to choose the degree of regularization γ, and fit the ℓ_1-regularized model.

annotation data set	variance w_1	points w_2	β
original	1.01	0.96	−2.66
	±0.03	±0.02	±0.10
detailed.low.density	1.30	0.93	−2.00
	±0.02	±0.02	±0.13
detailed.high.density	3.16	0.08	6.54
	±0.38	±0.26	±2.38

Table 5.2: Coefficients of the log.s.log.d model (5.29) were estimated using max-margin interval regression on several annotated data sets. Data sets were split into 10 folds, and we report the mean and standard deviation of coefficients learned across the 10 folds.

Change-point detection accuracy

We used cross-validation to compare the four models, and the test annotation error is shown in Figure 5.9. The log.d model that minimizes the surrogate loss shows comparable performance to cghseg.k. The log.s.log.d model shows change-point detection performance comparable to the ℓ_1-regularized model, but with much less training time. The step that takes a long time when training the ℓ_1 regularized model is the internal cross-validation loop that is used to select γ. So the 2 features suggested by the theory of Lavielle allow fast learning of a penalty function for change-point detection that performs just as well as the 117-dimensional ℓ_1-regularized model.

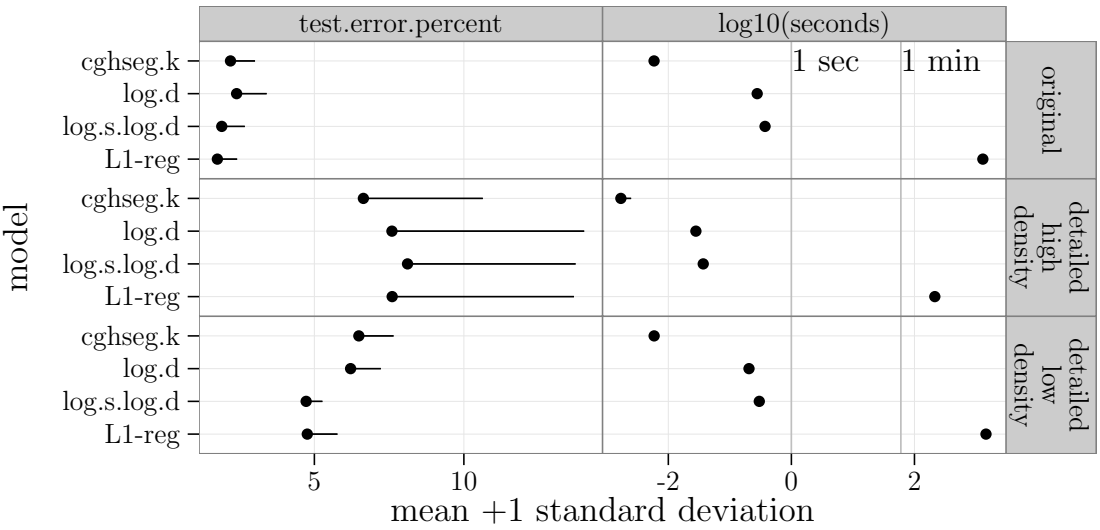

Figure 5.9: Breakpoint detection models were evaluated using 10-fold cross-validation. Percent test error is shown on the left and training time is shown on the right. The baseline cghseg.k model uses one-dimensional grid search to find a parameter that minimizes the annotation error. The log.d model uses interval regression on one feature $x_i = \log d_i$, and the log.s.log.d model uses two features $x_i = [\log d_i, \log \sigma_i]$. The L1-reg model many features $x_i \in \mathbb{R}^{117}$ and the ℓ_1 penalty with the degree of regularization γ chosen using an internal cross-validation step.

5.6 Conclusions

We proposed a method to learn the penalty function for change-point detection using annotated regions. This led us to an interval regression problem, which we solved using FISTA. We showed that learning the penalty function using this method results in state-of-the-art breakpoint detection performance in a large database of annotated copy number profiles.

This approach optimizes the shape of the penalty function term based on the annotated profiles. This way one can tune the parameters of her favorite model selection criterion in a reproducible manner, and avoid the use of heuristics which often yield suboptimal change-point detection.

To continue this line of work, we are considering learning more general penalty functions. For example, Lebarbier [2005] proposed a model complexity of the form $k(c_1 log(d_i/k) + c_2)$ and calibrated $c_1 = 2$ and $c_2 = 5$ using a large set of simulated profiles. It is reasonable to think that these values of c_1 and c_2 are not optimal for real data and one would like to learn these c_1, c_2 from a database of annotated profiles. To do so we are exploring multi-dimensional interval regression.

Chapter 6

Conclusions and future work

To summarize part 1 of this thesis, we have discussed several new methods for segmentation and clustering.

In Chapter 2 we introduced the clusterpath, which learns a tree structure from data by solving a sequence of convex optimization problems. This tree can then be cut to obtain discrete clusters. We compared several common clustering methods by assuming a number of desired clusters, and the clusterpath showed good cluster recovery in several situations. There are several directions for future research:

- In real data analyses, the number of clusters is generally unknown. Model selection criteria should be developed in order to automatically choose the clusterpath regularization parameter λ which controls the number of clusters.

- We proved that the ℓ_1 clusterpath is agglomerative when we use the identity weights, and we observed no splits using identity weights with the ℓ_2 clusterpath. To construct more efficient algorithms that do not need to check for splits, we should try to find conditions on the weights that guarantee no cluster splits.

In Chapter 3, we turned to the segmentation problem, which can be seen as a structured, linear version of clustering. In particular, choosing the number of clusters is often as difficult as choosing the number of segments or change-points in a segmentation model. Rather than using traditional model selection criteria, we proposed to select the model and number of segments that maximizes agreement with a database of breakpoint annotations. We also compared the breakpoint detection performance of several models on a set of 575 annotated copy number profiles of neuroblastoma tumors. We noted several lines of further research:

- Standard model selection criteria such as the BIC and mBIC showed inferior breakpoint detection performance to methods with parameters learned using the breakpoint annotations. Some theoretical work should be done to explain why this is the case, and to develop better criteria.

- We observed that different penalty parametrizations such as flsa and flsa.norm showed different breakpoint detection performance. This suggested that we should look for an optimal penalty that results in the best breakpoint detection. So in Chapter 4, we assumed several simple noise models and derived optimal penalties. And in Chapter 5, we proposed to learn in optimal penalty from databases of breakpoint annotations.

- We showed that relatively few annotations are needed to get a model with good breakpoint detection performance. This analysis was based on picking several annotations at random, as if we annotated a randomly selected subset of signals. However, there may be active learning strategies that can be used to select signals for annotation, and more quickly get a model with good breakpoint detection. For example, some signals clearly have no breakpoints and so once a few of these are annotated, we should try to annotate other signals with breakpoints. We have developed some preliminary active learning algorithms for this, but we need to develop methods for proving their effectiveness.

- We developed SegAnnDB, a web site for supervised, interactive breakpoint detection. Different experts can annotate the same signals, and thus potentially see different models of the same signal. An interesting direction for future research is to learn a breakpoint detection models based on sharing information from several different experts.

- We analyzed the 575 annotated copy number profiles in the neuroblastoma data set, but it will be interesting to apply annotation-based smoothing model selection to copy number profiles from other tumor types, and data types. Although we found pelt.n and cghseg.k to be the best for breakpoint detection, another model may be the best in a data set from another microarray technology or tumor type.

- We could use visual annotation databases to train copy number calling models. For example, SegAnnDB allows annotation of copy number status as normal, gain, or loss. These annotations could be used to

compare the performance if several copy number calling models, and also to train the parameters of these models.

- We could also use visual annotations in many other kinds of statistical models. In fact, visual annotations should be useful whenever a model is plotted on top of a data set. Whenever we examine a model in the context of its data, we are judging the goodness of the model fit. Often, it is possible to draw on the plot to indicate what a good or bad model would look like. For example, in clustering models, it should be possible to project the data onto a 2D subspace, then plot the data and identify pairs of points that should or should not be clustered together. These pairwise annotations can be used to select the optimal number of clusters, and one avenue for future research is to develop GUIs that implement this procedure.

In Chapter 4, we attacked the problem of developing an optimal penalty for breakpoint detection, using simulated noisy signals. We focused on the cghseg maximum likelihood Gaussian segmentation model, and discussed optimal penalties for number of points sampled, scale, and signal length in base pairs. There are several avenues of further research:

- We mathematically justified the scale normalization term σ_i^2 that we found empirically. But we did not offer a detailed theoretical analysis for the number of points normalization $\sqrt{d_i}$ that we found. This should be pursued, but may be difficult since the breakpoint error that we used to derive this result is not available in closed form.

- More generally, we focused on the breakpoint detection error, whereas most other model selection criteria focus on recovering the latent signal. We showed in Figure 4.3 an example signal for which the best model for signal reconstruction is not the same as the best model for breakpoint detection. It will be interesting to explore the differences between these criteria, and to use finite-sample model selection theory to develop optimal penalties for breakpoint detection rather than signal recovery.

- Finally, we saw that the penalty suggested by our theoretical analysis and the analysis of simulations did not offer the best breakpoint detection performance on the neuroblatoma data. This is probably due to a noise structure which is more complicated than the homoscedastic Gaussian noise that we assumed. It will be interesting to analyze the distribution of this noise, and then develop theoretical arguments for the corresponding optimal penalties.

In Chapter 5, we proposed a method to learn an optimal penalty function for breakpoint detection in databases of annotated signals. Rather than considering a penalty function suggested by theoretical arguments, we took a pragmatic viewpoint and proposed to learn the best penalty function for a particular database of signals and annotations. There are several research directions to pursue:

- We considered log-linear penalty functions, whose coefficients can be interpreted as exponents of terms in the penalty function. Although this is very interpretable, it may offer breakpoint detection inferior to learning a general non-linear function. To learn non-linear penalty functions, we are considering kernelized versions of the interval regression problem.

- We proposed to use interval regression, which uses a linear model to predict a scalar model complexity parameter. However, some model selection criteria have constants that can not be easily learned in this framework. For example, Lebarbier [2005] proposed a penalty with constants c_1, c_2 which can not be learned using the method we proposed. To directly learn these constants from annotation data, we would need a general multi-dimensional description of the annotation error function. We could then propose a multi-dimensional relaxation, which may permit learning these constants using multi-dimensional interval regression algorithms.

- We solved the ℓ_1-regularized interval regression algorithm using FISTA with warm restarts, which gives an approximate regularization path. However, the problem we proposed has a piecewise linear regularization path, so we could develop a path-following homotopy algorithm to find the exact regularization path [Rosset and Zhu, 2007].

We have spent much of this thesis developing algorithms for optimal breakpoint detection, since breakpoints are important indicators in cancer prognosis [Janoueix-Lerosey et al., 2009]. Now that we have methods for detecting biologically relevant breakpoints, the most important avenue of future research will be to construct prognostic models of patient outcome, based on the detected breakpoints, gains, and losses in their tumors.

Part II

Statistical software contributions

Chapter 7

Adding direct labels to plots

Some content of this chapter is taken from my poster, "Adding direct labels to plots," which won the Best Student Poster award at useR 2011, the international R conference which took place at the University of Warwick, England.

Chapter summary

This chapter discusses **directlabels**, an R package for adding direct labels to multicolor plots. Examples of direct labels can be seen in this thesis in Figures 3.3, 3.4, which are reproduced here in Figure 7.1. Direct labels are often much easier to interpret than a legend, but are not often used in practice since the label positions must be determined based on the data.

More specifically, the topic of this chapter is the design of algorithms for direct label placement. Manually specifying direct label positions for plots is always possible, and often done in practice to polish figures for publication. However, statisticians interpret many multicolor plots when performing exploratory data analysis. Manually determined direct labels are not often used for these plots since it takes too much time to determine the label positions.

The **directlabels** package provides algorithms for automatic direct label placement that can be used in everyday data analysis. This chapter describes:

- Algorithms that can be used to find readable direct labels for several plot types.
- **directlabels**, the R package that implements these algorithms for use with **grid** graphics plotting systems such as **lattice** and **ggplot2** [Sarkar, 2008, Wickham, 2009].

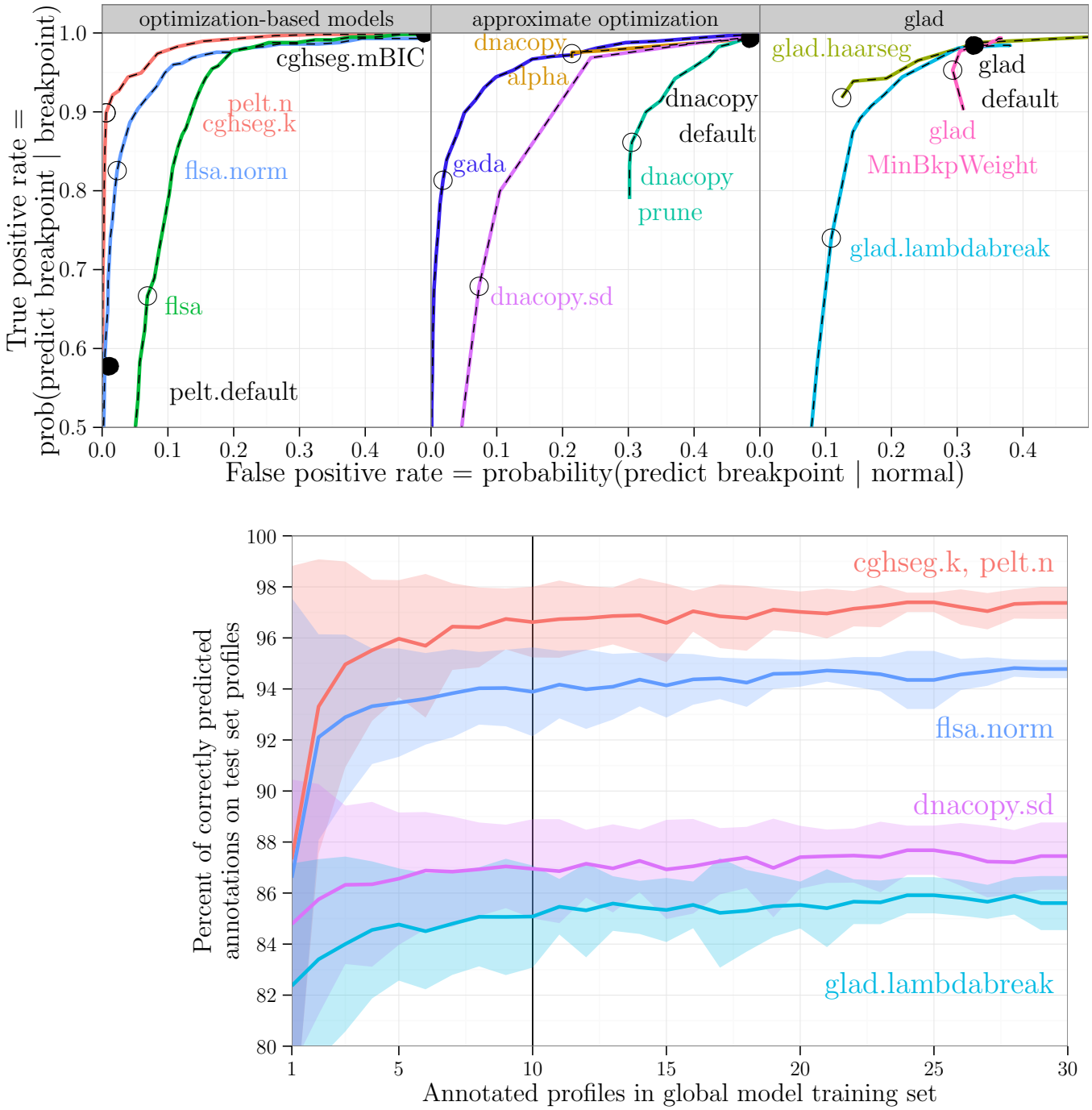

Figure 7.1: Some examples of direct labels in graphics, taken from Figures 3.3 and 3.4 earlier in this thesis. Direct labels show the meaning of the different colors, which correspond to different breakpoint detection models in these plots. Direct labels are a readable alternative to legends since their positions are determined as a function of the data.

7.1 Introduction and related work

Direct labels are not a new technique for data visualization. In fact, Tufte [2001] encourages statisticians to label their plots via manual visual label placement. This approach is always possible with GUI tools, but becomes impossible for the analysis of automatically-generated plots. Multicolor plots are often labeled automatically using legends, which can be difficult to read. The purpose of this work is to introduce new algorithms for automatic direct label placement that result in plots that are easy to read.

In his book on R Graphics, Paul Murrell of the R core development team offers a review of plot labeling systems for R, and notes that "the **directlabels** package, still in development at the time of writing, provides the `direct.label()` function for a general labeling paradigm with **grid**-based packages" [Murrell, 2011, page 347].

That review also mentions several other functions for direct labeling, including `spread.labels`, `thigmophobe.labels`, `spread.labs`, `pointLabel`, and `labcurve`. The `labcurve` function is made for labeling lineplots, which we discuss in Section 7.3. The other functions are for labeling *individual points* on scatterplots, which is in general an NP-hard problem related to automatic label placement on maps [Wikipedia, 2012b]. In contrast, legends are used for labeling *groups of points* of different colors, and the **directlabels** package proposes to replace confusing legends with direct labels.

There is also a technical distinction: these other functions work with R base graphics, whereas **directlabels** works with R grid graphics. For a review on the differences between these two systems, read the textbook of Murrell [2011, Chapters 2 and 6]. In principle, the direct labeling algorithms discussed in the next sections could be used with R base graphics. However, the **directlabels** package was designed to exploit the structure of **lattice** and **ggplot2** plots, which are implemented using grid graphics.

In the next sections, we discuss methods for direct labeling densityplots, lineplots, and scatterplots. Then, at the end of the chapter, we discuss the implementation of these methods in the **directlabels** package.

7.2 Densityplot labels

Densityplots are visualizations that attempt to summarize univariate data distributions. They are particularly useful for comparing distributions. For example, Figure 7.2 shows a densityplot that compares 3 data distributions, labeled using different colors.

Look carefully at the legend in the plot on the top of Figure 7.2. It is apparent that the data distribution does not match the legend, which was laid out in alphabetical order. This plot could easily be misinterpreted if it was printed in black and white.

To solve this problem, we need to use direct labels, which use the data to determine the label positions. In the bottom half of Figure 7.2, we place direct labels at the mode of each density curve. In conclusion, simply replacing an alphabetically-ordered legend with data-based direct labels results in a plot that is much easier to interpret.

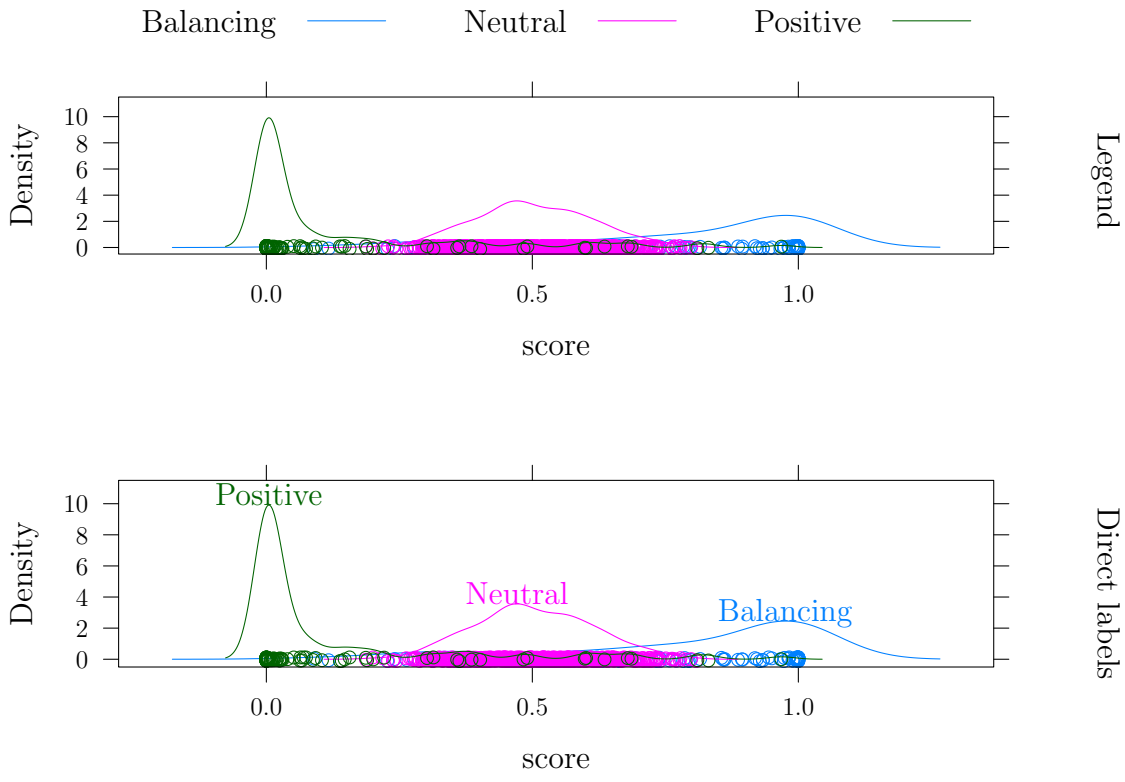

Figure 7.2: **Top**: densityplot with an alphabetically-ordered legend that is inconsistent with the data. **Bottom**: adding direct labels at the mode of each density results in a plot that is much easier to interpret.

7.3 Lineplot labels using a quadratic program

To find a readable direct labeling algorithm for lineplots, let us consider the model shown in the left panel of Figure 7.3. In particular, assume that for each text label $k \in \{1, \ldots, K\}$ we have its target position $t_k \in \mathbb{R}$ and height $h_k \in \mathbb{R}^+$. We will develop several different methods for defining the optimal position y_k^* for each label.

The naïve method defines $y_k^* = t_k$, centering each label at the end of its line. This method is depicted in the left panel of Figure 7.3, in which two rectangles overlap so the labels would be unreadable. For a real example, consider the top of Figure 7.4, which is taken from Figure 4.4 earlier in this thesis. The naïve method indeed sometimes results in overlapping, unreadable labels.

To solve the problem of overlapping labels, we formulate lineplot labeling as a convex optimization problem. For a good introduction to convex optimization, see Boyd and Vandenberghe [2004]. We would like to find direct label positions y_k that do not overlap, and this can be written using these linear inequalities:

$$y_{k+1} - h_{k+1}/2 \geq y_k + h_k/2, \ \forall \ k \in \{1, \ldots, K-1\}. \tag{7.1}$$

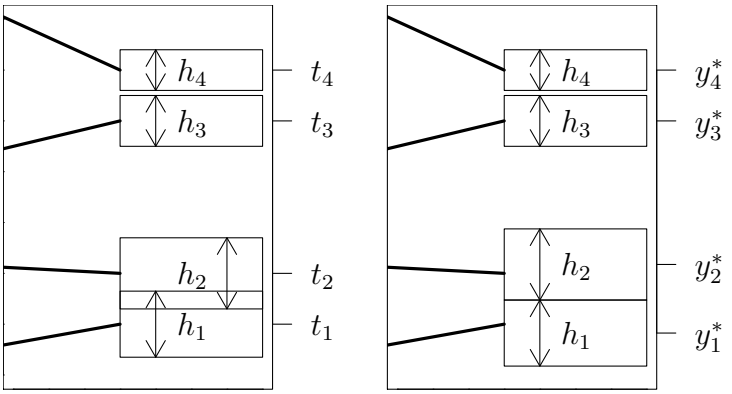

Figure 7.3: **Left**: we model lineplot labels using their text bounding boxes, shown as rectangles. The target position t_k of each label is where the bounding box is centered around its line (bold), and the height h_k of the box is known (arrows). In this example, labels 1 and 2 overlap, so will be difficult to read. **Right**: after applying the QP solver, we obtain the optimal positions y_k^*, which do not overlap.

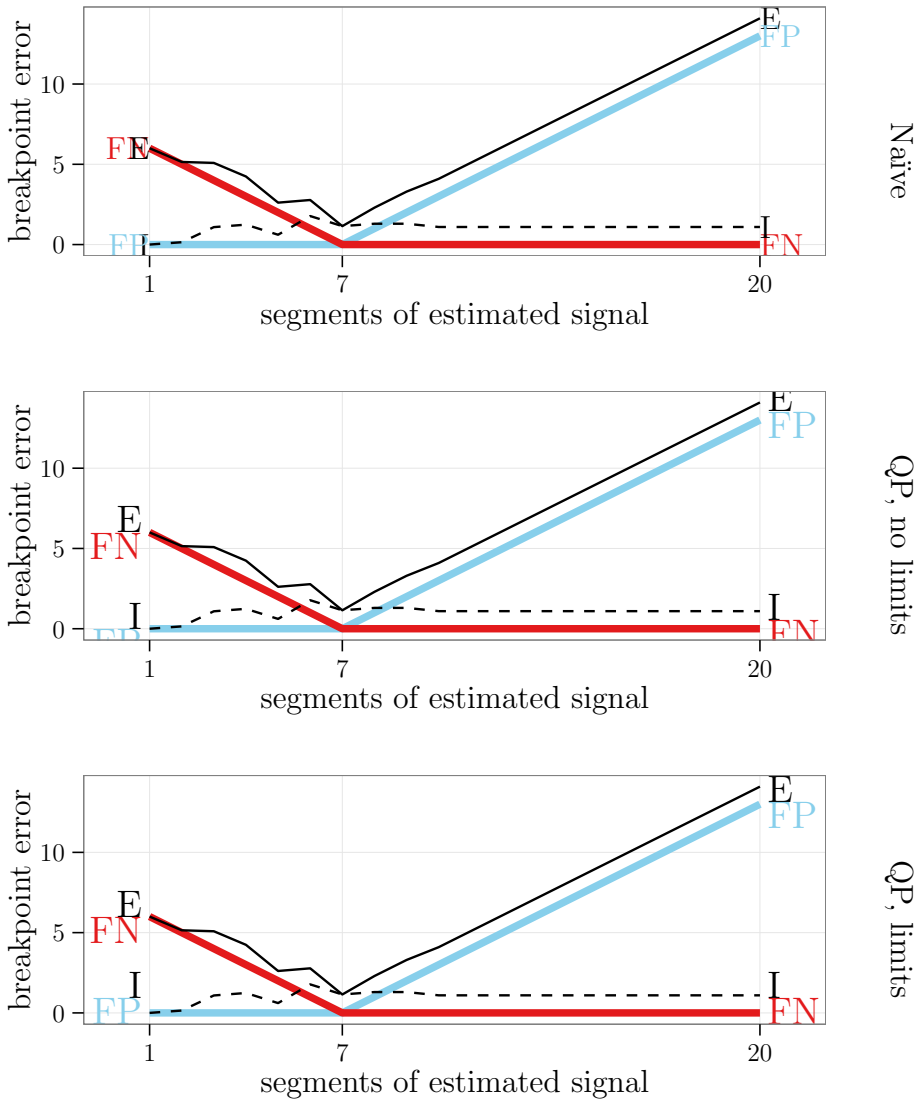

Figure 7.4: **Top**: the naïve labeling strategy simply places each label at its line, and can result in unreadable, overlapping labels. **Middle**: a quadratic program (QP) is used to find labels that do not overlap, but go out of the plotting region so are only partially readable. **Bottom**: adding constraints on the top and bottom labels results in labels that stay in the plotting region.

We would like to find label positions y which are as close as possible to the target locations t, suggesting the optimization problem

$$\begin{aligned} \min_{y \in \mathbb{R}^K} \quad & \sum_{k=1}^{K} (y_k - t_k)^2 = ||y - t||_2^2 \\ \text{subject to} \quad & y_{k+1} - h_{k+1}/2 \geq y_k + h_k/2, \ \forall \ k \in \{1, \ldots, K-1\}. \end{aligned} \tag{7.2}$$

This is a quadratic program (QP) since the objective is quadratic and the constraints are linear. Furthermore, the objective is strongly convex with constant 1, which means there is a *unique* solution which corresponds to the best labels [Boyd and Vandenberghe, 2004, Chapter 9].

We can solve problem (7.2) using the dual method of Goldfarb and Idnani [1983], which is implemented in the `solve.QP()` function in the **quadprog** R package [Turlach and Weingessel, 2011]. To use the solver, we must write the QP in standard form:

$$\min_{y \in \mathbb{R}^K} \quad \tfrac{1}{2} y'y - t'y \tag{7.3}$$

$$\text{subject to} \quad A'y = \begin{bmatrix} -1 & 1 & 0 & & & \\ 0 & -1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & -1 & 1 & 0 \\ & & & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix} \geq \underbrace{\begin{bmatrix} (h_1 + h_2)/2 \\ \vdots \\ (h_{K-1} + h_K)/2 \end{bmatrix}}_{\texttt{(h[-K]+h[-1])/2}}$$

where A is the $K \times K-1$ constraint coefficient matrix:

$$A = \begin{bmatrix} -1 & 0 & & & \\ 1 & -1 & & & \\ 0 & 1 & \ddots & & \\ & & \ddots & -1 & 0 \\ & & & 1 & -1 \\ & & & 0 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & \cdots & 0 \\ 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}}_{\texttt{rbind(0,}I_{K-1}\texttt{)}} - \underbrace{\begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \\ 0 & \cdots & 0 \end{bmatrix}}_{\texttt{rbind(}I_{K-1}\texttt{,0)}} \tag{7.4}$$

This method gives the direct labels shown in the middle panel of Figure 7.4, which indeed prevents the labels from overlapping. However, some of the labels appear outside the plotting region, so are unreadable.

In general, we do not want the labels to go outside the top $\overline{L}$ and bottom $\underline{L}$ limits of the plotting region. So to avoid this, we can impose $y_1 - h_1/2 \geq \underline{L}$ and $y_K + h_K/2 \leq \overline{L}$, which translate into the following standard form constraints:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & -1 \end{bmatrix} y \geq \begin{bmatrix} \underline{L} + h_1/2 \\ h_k/2 - \overline{L} \end{bmatrix} \tag{7.5}$$

Adding these constraints to QP (7.3) results in the direct labels shown on the bottom of Figure 7.4. So using quadratic programming, we get direct labels for lineplots that are constrained to be readable.

As another example of the utility of this method, consider the lineplot on the top of Figure 7.5. The legend in this plot attempts to show the correspondence between color and ID, but is unreadable for two reasons. First, there are simply too many entries in the legend, so it is impossible to map colors back to IDs. Second, the legend entries are ordered using the value of an unplotted variable.

We can use the same QP to obtain the readable direct labels shown on the bottom of Figure 7.5. In addition, the direct labels show the ordering of the ID variable with respect to the plotted weight variable.

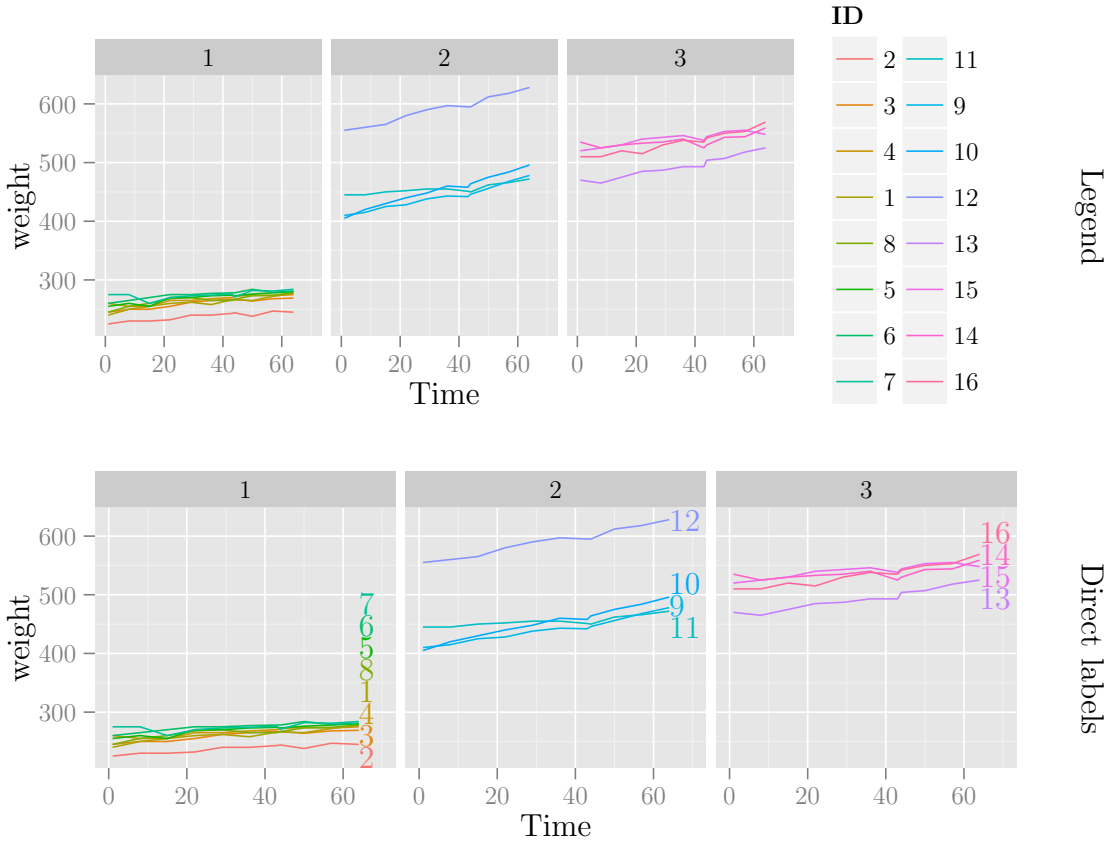

Figure 7.5: **Top**: too many classes render a legend unreadable. **Bottom**: direct labels are readable and show the class order.

Finally, let us consider direct labels for lineplots of Lasso coefficients. We use the prostate data set described by Hastie et al. [2009, Figure 3.10]. The naïve label positions for Lasso coefficients are where the variable enters the path, and at the end of the lines, as shown in the top panel of Figure 7.6. These labels can sometimes overlap, and so are not easy to read.

To solve the problem of overlapping labels at the end of the lines, we can directly use the solutions of (7.3) obtained from the QP solver. The other labels are rotated at an angle of $\theta \in [0, 2\pi]$, so the width of each label is $h_k / \sin(\theta)$, where h_k is the height of un-rotated label k. To apply the QP solver, we substitute $h_k / \sin(\theta)$ for h_k in problem (7.3), and define the target t_k as the horizontal position where line k enters the path. This method gives the clear, readable labels shown in the bottom panel of Figure 7.6.

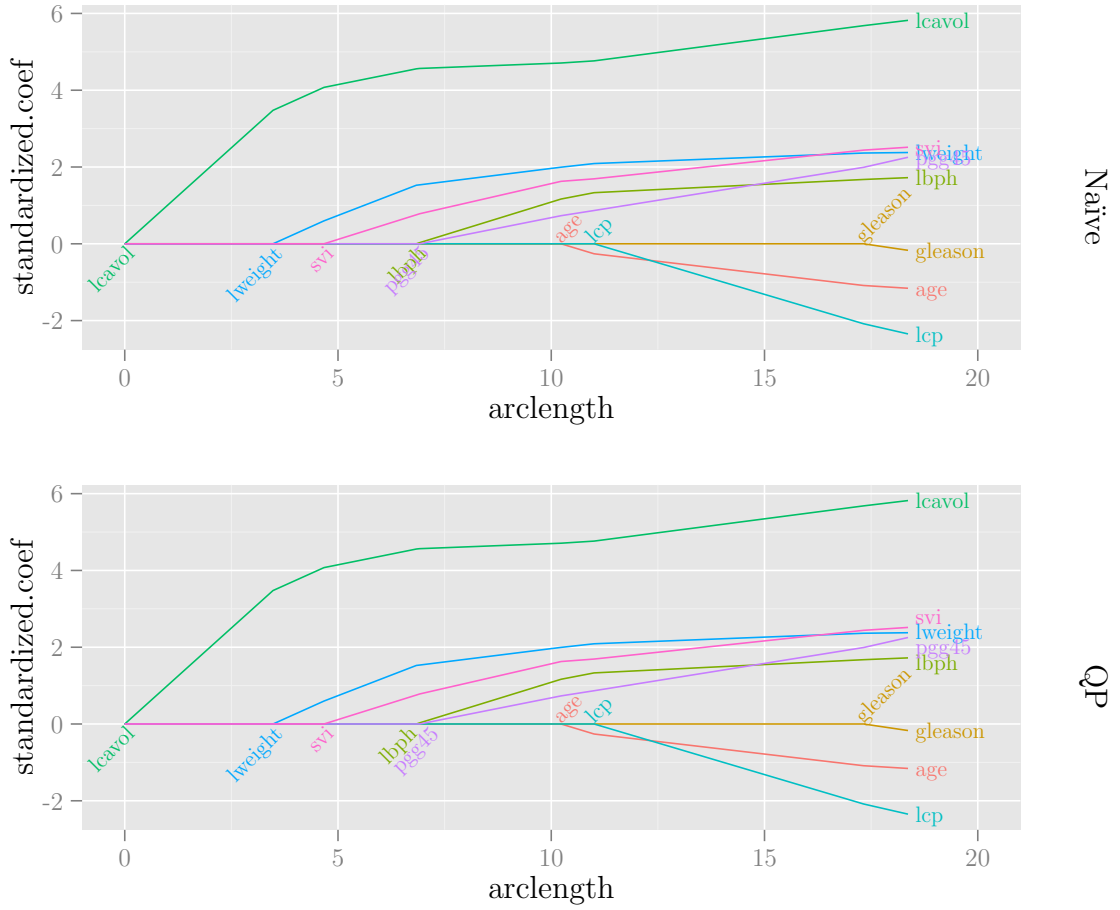

Figure 7.6: **Top**: naïve direct labels for Lasso coefficients can sometimes overlap. **Bottom**: the QP solver can be used to avoid overlaps.

7.4 Scatterplot labels

Finding the optimal direct labels for scatterplots is a problem closely related to automatic label placement on maps. There are rule-based, greedy, simulated annealing, and other optimization algorithms for this task [Wikipedia, 2012b].

To define the labeling problem, take the scatterplot of the iris data in the top panel of Figure 7.7 as a motivating example. There are $K = 3$ classes labeled using a legend and different colored points on the scatterplot. The goal of this section is to define the position $L_k \in \mathbb{R}^2$ of the label for each class $k \in \{1, \ldots, K\}$.

There are several properties that we would like for scatterplot label positions:

- **(close to class center)** Each label appears close to the center c_k of its point cloud, for easy decoding.
- **(no label-point overlap)** Labels do not overlap any points, so are readable.
- **(no label-label overlap)** Labels do not overlap other labels, so are readable.

These criteria suggest the optimization problem

$$\begin{aligned} \min_{L \in \mathbb{R}^{K \times 2}} \quad & \sum_{k=1}^{K} ||L_k - c_k||_2^2 \\ \text{subject to} \quad & B_k(L_k) \cap H = \emptyset \ \forall k \in \{1, \ldots, K\} \\ & B_k(L_k) \cap B_j(L_j) = \emptyset \ \forall k \neq j, \end{aligned} \tag{7.6}$$

where the notation is as shown in the bottom panel of Figure 7.7:

- $L_k \in \mathbb{R}^2$ is the position of label k.
- $c_k \in \mathbb{R}^2$ is the center of point cloud k.
- $B_k : \mathbb{R}^2 \to 2^{\mathbb{R}^2}$ is the bounding box of label k.
- $H \subset \mathbb{R}^2$ is the hull of data points, which can be defined in many ways.

Although the objective function in problem (7.6) is convex, the constraints are non-convex. So any optimization algorithm that uses gradient optimality is guaranteed to find only a local minimum.

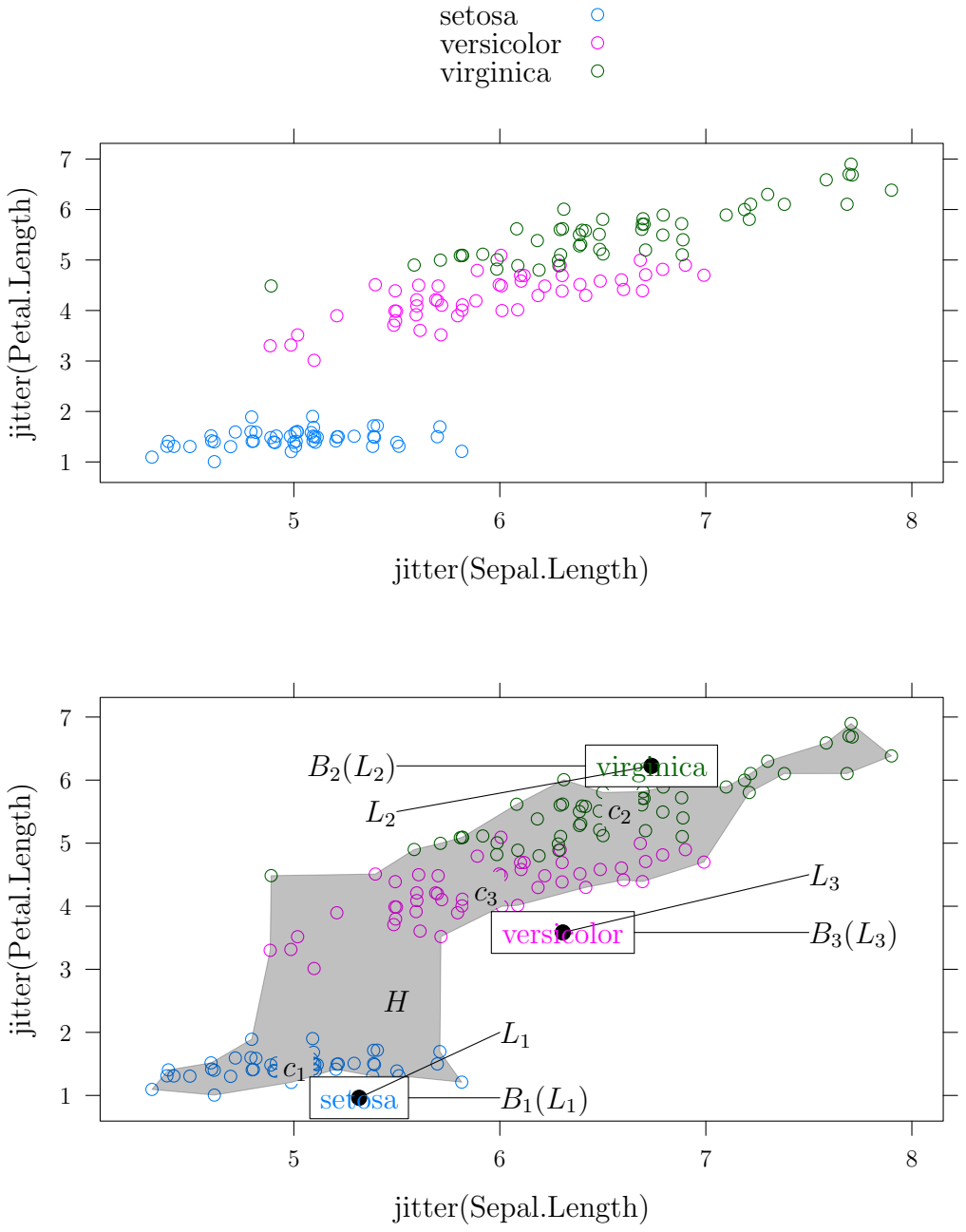

Figure 7.7: **Top**: scatterplot of the iris data with legend for decoding. **Bottom**: direct label positions L_k are drawn as black dots, bounding boxes B_k are drawn as rectangles, point cloud centers c_k are shown, and the hull H of data is shaded in grey.

So we propose the following algorithm based on grid search to find an approximate solution to problem (7.6). For every point $j \in \{1, \ldots, n\}$, let $x_j \in \mathbb{R}^2$ denote its position on the scatterplot, and let $\kappa_j \in \{1, \ldots, K\}$ be its class.

- Initialize the hull of points used for overlap detection to the set of plotted points
$$H = \{x_1, \ldots, x_n\}. \tag{7.7}$$
- For each class $k \in \{1, \ldots, K\}$:
  1. Let $J_k = \{j : \kappa_j = k\}$ be the indices of the points in this class.
  2. Calculate its center $c_k \in \mathbb{R}^2$ using the mean:
$$c_k = \frac{1}{|J_k|} \sum_{j \in J_k} x_j. \tag{7.8}$$
  3. Calculate the size of text label k to obtain a bounding box B_k.
  4. Use the text label size to define a grid of possible label positions $G \subset \mathbb{R}^2$ over the entire plot.
  5. For each grid point $g \in G$, define a function $\varphi : \mathbb{R}^2 \to \{1, \ldots, n\}$ that gives the index of the nearest data point:
$$\varphi(g) = \underset{j \in \{1,\ldots,n\}}{\arg\min} ||g - x_j||_2^2. \tag{7.9}$$
  6. Define the feasible points F as the set of grid points g closest to class k which contain no points in a bounding box $B_k : \mathbb{R}^2 \to 2^{\mathbb{R}^2}$ around g:
$$F = \{g \in G \text{ such that } \varphi(g) \in J_k \text{ and } B_k(g) \cap H = \emptyset\}. \tag{7.10}$$
  7. Place the label at the nearest feasible grid point:
$$L_k = \underset{g \in F}{\arg\min} ||g - c_k||_2^2. \tag{7.11}$$
  8. Add some points in $B_k(L_k)$ to the set of points H used for overlap detection.

This algorithm is illustrated for the iris data in Figure 7.8. By construction, this algorithm gives label positions that obey the non-overlapping constraints. However, although the label positions L are the closest points in F to the class centers c, they are not a local minimum of problem (7.6). Gradient-based methods could be used to find a local minimum, but in practice the grid search suffices to find readable scatterplot labels.

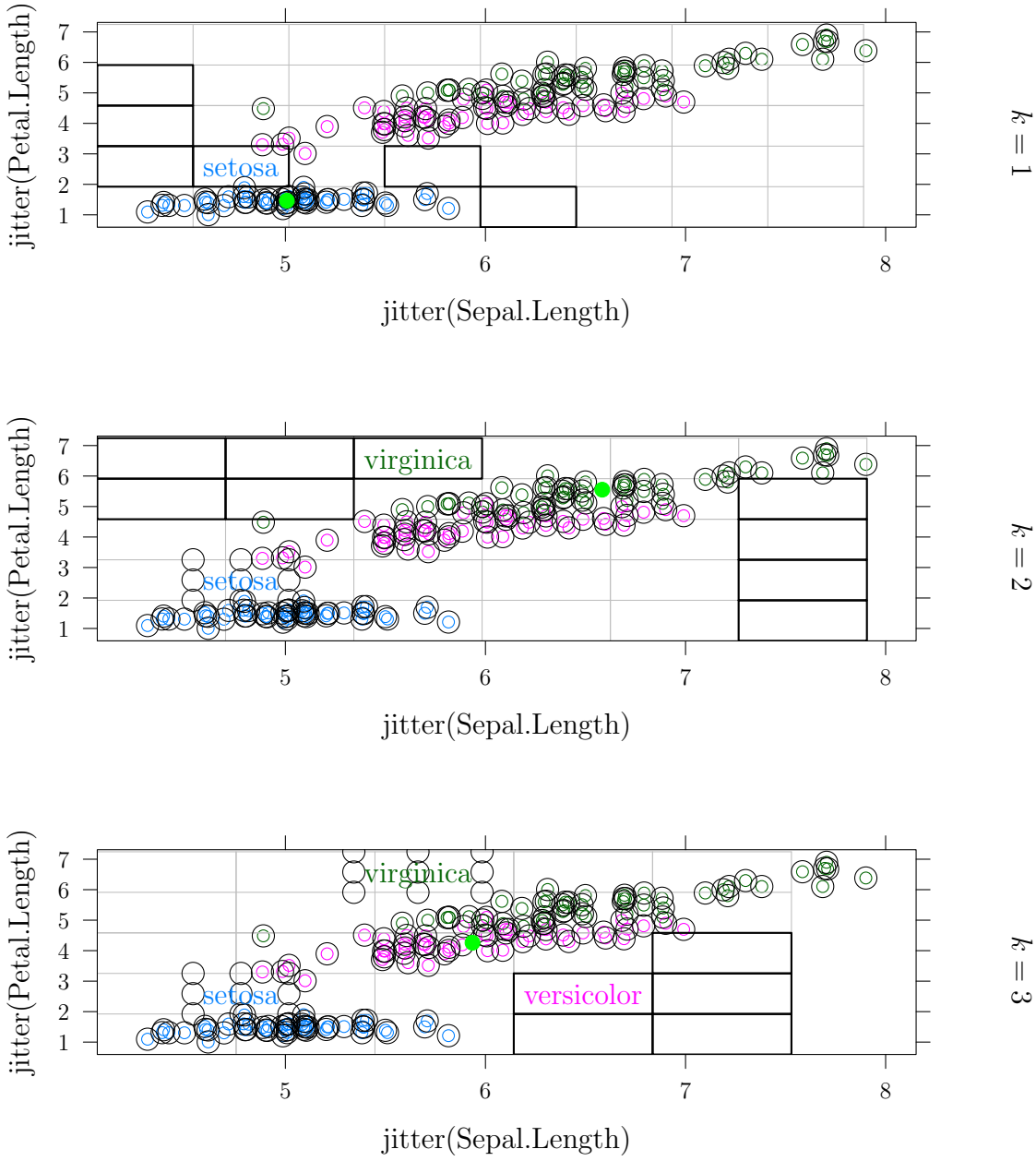

Figure 7.8: Illustration of the algorithm for scatterplot labeling based on grid search. The whole grid G is drawn as grey rectangles, and the feasible set F is highlighted with black rectangles. The set of points H used to avoid overlaps is drawn in black, and the class centers c_k are drawn in green. Note in the middle and bottom panels that 9 points are added to the set H based on the label bounding box $B_k(L_k)$ found in the previous step.

7.5 Design of directlabels

In this section, I will briefly explain the design of the **directlabels** package. These were the design goals:

- Adding direct labels should require very little code, to make it practical for real data visualization.

  So, the `direct.label` function includes various heuristics to infer sensible direct label positions for common plot types. As shown on the top of Figure 7.9, `direct.label(Plot)` takes a `Plot` object and returns another `DirectLabeledPlot` object.

- Label placement methods can be used with both **lattice** and **ggplot2**.

  To address this point, the `direct.label` function is generic, as shown in the middle of Figure 7.9. The **directlabels** package defines methods for `trellis` objects from the **lattice** package, and `ggplot` objects from the **ggplot2** package. Thus, adding direct labels to a `Plot` object is accomplished using `direct.label(Plot)` for both types of objects.

- Defining methods for label placement should be straightforward.

  To address this last point, label positions are defined using Positioning Methods. A Positioning Method is an R `function(d,...)`, where `d` is a data.frame with columns `x`, `y`, and `groups` that define the points to plot. The job of the Positioning Method is to return a data.frame of direct label positions, as shown in the bottom of Figure 7.9.

Finally, note in the bottom of Figure 7.9 that the **directlabels** package creates a `dlgrob`, a special type of graphical object that plots direct labels. A graphical object or `grob` is essentially a list with data to be plotted by the **grid** package. Reference on grobs and other **grid** graphics concepts is given by Murrell [2011, Chapter 6].

The job of a `dlgrob` is to call the Positioning Method with the plot region as the current **grid** viewport. This means that inside of a Positioning Method, it is possible to determine the

- Plot region limits $\overline{L}, \underline{L}$ using the **grid** function `convertY`.

- Label height h_i using functions `stringHeight` and `convertHeight`.

Thus, it is possible to use this information to implement the labeling algorithms for lineplots and scatterplots proposed in Sections 7.3 and 7.4.

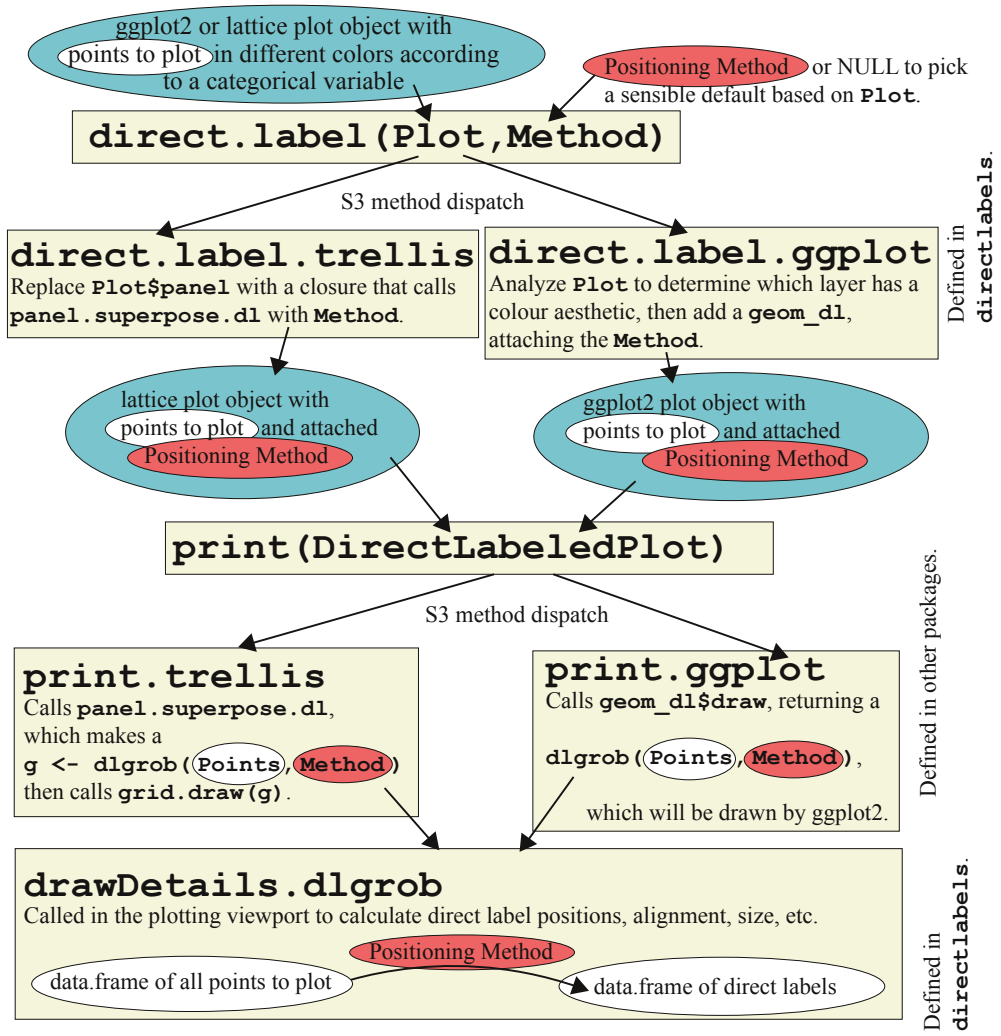

Figure 7.9: The design of the **directlabels** package. Functions are drawn as beige rectangles, `Plot` and `DirectLabeledPlot` objects are drawn using teal ovals, data.frames of points and labels are drawn as white ovals, and the Positioning Method is drawn using a red oval.

7.6 Conclusions

In this chapter, we have discussed several algorithms for adding direct labels to common plot types. Using the implementation of these algorithms in the R package **directlabels**, we have shown examples of readable direct labels for several data sets.

Furthermore, we have argued that the **directlabels** package allows use of direct labels as an alternative to legends in automatically-generated statistical plots. It is clear that manual definition of label positions is preferable for publication-quality statistical graphics. However, **directlabels** should be useful to quickly obtain readable direct labels in everyday data analysis.

Chapter 8

Sustainable, extensible documentation generation using inlinedocs

This material from this chapter is taken from the **inlinedocs** article, and is joint work with Thomas Wutzler, Keith Ponting, and Philippe Grosjean [Hocking et al., in press]. R is a statisical programming language that I used to implement many of the methods described in the previous chapters. Code written in the R language is normally distributed in a standard form called a "package," which also contains documentation that describes the code. This chapter describes **inlinedocs**, a new system for writing that documentation. I used **inlinedocs** to write the documentation for the R packages mentioned in this thesis: **bams**, **clusterpath**, **directlabels**, and **inlinedocs** itself.

Chapter summary

This chapter presents **inlinedocs**, an R package for generating documentation from comments. The concept of structured, interwoven code and documentation has existed for many years, but existing systems that implement this for the R programming language do not tightly integrate with R code, leading to several drawbacks. This chapter attempts to address these issues and presents 2 contributions for documentation generation for the R community. First, we propose a new syntax for inline documentation of R code within comments adjacent to the relevant code, which allows for highly readable and maintainable code and documentation. Second, we propose an extensible system for parsing these comments, which allows the syntax to be easily augmented.

8.1 Introduction

In this chapter, we present **inlinedocs**, an R package which allows R documentation to be written in comments. The standard way to distribute R code is in a package along with Rd files that document the code [R Development Core Team, 2012]. There are several existing methods for documenting a package by writing R comments, which are later processed and converted into standard Rd files. We first review these efforts, emphasizing the key issues that justify the introduction of a new package like **inlinedocs**.

Existing documentation generation systems for R

For report generation and literate programming, the mature Sweave [Leisch, 2003] format allows integration of R code and results within LaTeX documents [Lamport, 1986]. However, the goal of **inlinedocs** is different. It aims for integration of documentation inside of R code files, to generate Rd files using R code and markup in R comments. Thus for **inlinedocs** we need to extract the documentation specified in R code, and the Sweave system can not be easily applied to this parsing task.

The `package.skeleton` function that ships with base R is intended to ease the generation of Rd files from R code. After specifying some input R code files or objects to use for the package, it produces some minimal documentation that must be completed using a text editor. Although `package.skeleton` is sufficient for creating small packages that are published once and forgotten, it offers little help for continued maintenance of packages for which Rd files are frequently updated.

The other existing approaches, **Rdoc** [Bengtsson, 2010] and **Roxygen** [Danenberg, 2009], attempt to address this sustainability problem using Rd generation from comments in R code. The documentation is thus written closer to the code it documents, which is easier to maintain. These packages are a step toward seamless integration of code and documentation, but they have three major drawbacks:

1. They only use comments to generate documentation, ignoring the information already defined in the code. This is particularly problematic for documenting function arguments, which requires the repetition of the argument names in the function definition and the documentation. This repetition is a possible source of disagreement between code and documentation if both are not simultaneously updated.

2. The documentation for an object appears in comments above its definition. These comment blocks can grow to be quite large, and thus they tend to be far away from the relevant code.

3. Examples are defined either in comments or in supplementary R code files. Examples in comments are not easy to test and debug with the R interpreter, and supplementary R code files reintroduce the separation of code and documentation that these tools are supposed to eliminate.

There are many tools that accomplish documentation alongside code in other programming languages. Notable examples include docstrings in Lisp and Python, **Javadoc** for Java, and **Doxygen**, which supports several languages [Wikipedia, 2012a]. These systems use large comments in headers, and do not support R. In contrast, **inlinedocs** is designed for R packages, uses smaller comments alongside the code, and exploits the code structure to reduce the need to repeat information in the documentation.

Documentation using inline comments

The **inlinedocs** package addresses the aforementioned issues by proposing a new syntax for inline documentation of R packages. Using **inlinedocs**, one writes documentation in comments right next to the relevant code, and examples in the `ex` attribute of the relevant object. By design, **inlinedocs** exploits the structure of the R code so that only minimal documentation comments are required, reducing duplication and simplifying code maintenance.

The remainder of this chapter is organized as follows. In Section 8.2, we discuss the details of the **inlinedocs** syntax for writing documentation in R comments. In Section 8.3, we discuss the design and implementation of **inlinedocs**, and explain how the syntax can be extended. In Section 8.4, we conclude and offer some ideas for future improvements.

8.2 The inlinedocs syntax for inline documentation of R packages

The main idea of **inlinedocs** is to document an R object using `###` and `##<<` comments directly adjacent to its source code. Furthermore, **inlinedocs** allows documentation wherever it is most relevant in the code using `##section<<` comments. These special comment strings are designed to work well with the default behavior of common editing environments, such

as **Emacs** with the **Emacs Speaks Statistics** [Rossini et al., 2004] add-on package:

- `###` is aligned to the left margin, providing maximum space for comment text.
- `##<<` is aligned with the start of adjacent code lines, so that comments using this form in the middle of a function do not obscure the code structure.

The following Sections illustrate common usage of **inlinedocs** comments through **fermat**, an example package inspired by the **Roxygen** vignette [Danenberg, 2009]. The examples were processed and checked for validity using **inlinedocs** version 1.9. For brevity, only the most frequently used **inlinedocs** features will be discussed, and the reader is directed to the **inlinedocs** web site for complete documentation:

`http://inlinedocs.r-forge.r-project.org/`

Documenting function arguments and return values

The following example demonstrates the minimal documentation a package author should provide for every function. Note that the location of white space, brackets, default arguments and commas is quite flexible.

```
fermat.test <- function
### Test an integer for primality using Fermat's Little Theorem.
(n ##<< The integer to test.
 ){
  a <- floor(runif(1,min=1,max=n))
  a^n %% n == a
### Whether the integer passes the Fermat test for a randomized
### \eqn{0<a<n}.
}
```

The comments correspond to the following sections of the `fermat.test.Rd` file:

- `###` comments following the line of `function` form the `description` section.
- For each argument, an item is created in the `arguments` section using a `##<<` comment on the same line.
- `###` comments at the end of the function form the `value` section.

By default, `name`, `alias` and `title` Rd sections are set to the function name, so this minimal level of documentation is enough to make a working package that passes `R CMD check` with no errors or warnings.

Inline titles, arguments, and other sections

The following example shows some optional **inlinedocs** comments that allow detailed and flexible specification of Rd files.

```
is.pseudoprime <- function # Check an integer for pseudo-primality.
### A number is pseudo-prime if it is probably prime, the basis of
### which is the probabalistic Fermat test; if it passes two such
### tests, the chances are better than 3 out of 4 that \eqn{n} is
### prime.
##references<< Abelson, Hal; Jerry Sussman, and Julie
##Sussman. Structure and Interpretation of Computer
##Programs. Cambridge: MIT Press, 1984.
(n,    ##<< Integer to test for pseudoprimality.
 times
### Number of Fermat tests to perform. More tests are more likely to
### give accurate results.
 ){
  if(times==0)TRUE
  ##seealso<< \code{\link{fermat.test}}
  else if(fermat.test(n)) is.pseudoprime(n,times-1)
  else FALSE
### logical TRUE if n is probably prime.
}
```

On the first line, the `#` comment specifies the title. On the lines after an argument, `###` comments specify its documentation. This is a useful alternative to inline `##<<` comments for longer, multi-line documentation of function arguments.

A `##section<<` comment can be used anywhere within a function, for any documentation section except `examples`, which is handled in a special manner as shown below in section 8.2. In each comment, arbitrary Rd may be written, as shown in the `##seealso<<` section above. Each `##section<<` may occur several times in the documentation for a single object. Such multiple occurrences are normally concatenated as separate paragraphs, but special processing is applied to match the intended use of the following documentation sections:

- `title` sections are concatenated into a single line.
- `description` sections should be brief, so are concatenated into a single paragraph.
- `alias` contents are split to give one alias per line of text.
- `keyword` contents are split at white space, each generating a separate `\keyword` entry.

The `###` and `##<<` documentation styles may be freely mixed. In general, `###` or `#` lines are processed first, followed by any corresponding `##<<` or `##section<<` comments. Section 8.3 will explain in more detail how comments are processed.

Examples and named lists

The following code demonstrates inline documentation of named lists, and the preferred method of writing examples:

```
try.several.times <- structure(function
### Test an integer for primality using different numbers of tests.
(n,    ##<< integer to test for primality.
 times ##<< vector of number of tests to try.
 ){
  is.prime <- sapply(times,function(t)is.pseudoprime(n,t))
  ##value<< data.frame with columns:
  data.frame(times, ##<< number of Fermat tests.
             is.prime, ##<< TRUE if probably prime
             n) ##<< Integer tested.
  ##end<<
},ex=function(){
  try.several.times(6,1:5)
  try.several.times(5,1:5)
})
```

On the final lines of the function definition, a `##value<<` comment allows documentation of lists or data frames using the names defined in the code. The entries are documented using `##<<` in the same way as function arguments, and this even works for nested lists. The `##end<<` comment closes the return value documentation block.

The examples are written using `structure` to put them in the `ex` attribute as the body of a function without arguments. This method for

documenting examples was motivated by the desire to express examples in R code rather than in R comments, to keep the examples close to the object definition, and to avoid repetition of the object name. When examples are in R code, they are easily transferred to the R interpreter, and thus are easy to debug. Furthermore, when examples are written close to the object definition, it is easy to keep examples up to date and informative.

An alternative is `attr(try.several.times,"ex") <- function(){`*code*`}` later in the code. However, we prefer using `structure` since it keeps the examples near the object definition, and avoids repetition of the object name.

The simplicity of adding examples and generating a package using **inlinedocs** also allows for routine regression testing of functions with very little extra work. Even for small collections of functions, one can use `R CMD check` to run the examples and optionally check the output with reference output.

Documenting classes and methods

S3 methods may be defined using plain R, or using `setConstructorS3` and `setMethodS3` from the **R.oo** package [Bengtsson, 2003]. The **inlinedocs** package detects S3 methods using `utils::getKnownS3generics` and `utils::findGeneric`, and updates the generated documentation automatically. S4 class declarations using the `setClass` function are also supported. The following example is from the source of **inlinedocs**:

```
setClass("DocLink", # Link documentation among related functions
### The \code{DocLink} class provides the basis for hooking together
### documentation of related classes/functions/objects. The aim is that
### documentation sections missing from the child are inherited from
### the parent class.
    representation(name = "character", ##<< name of object
                   created = "character", ##<< how created
                   parent = "character", ##<< parent class or NA
                   code = "character", ##<< actual source lines
                   description = "character") ##<< more details
         )
```

The inheritance referred to in this example is designed to avoid the need for repetitive documentation when defining a class hierarchy. The argument descriptions and other documentation sections default to those defined in the parent class. At present it only functions when all the definitions are within a single source file and this "documentation inheritance" is strictly linear within the file.

`package.skeleton.dx` for generating Rd files

The main function provided by **inlinedocs** is `package.skeleton.dx`, which generates Rd files for a package, and should be run before `R CMD build`. For example, `package.skeleton.dx("fermat")` processes R code found in `fermat/R`, and generates Rd files in `fermat/man` for each object in the package. Documentation is generated even for objects that are not exported. The generated Rd files should be treated as object files, since any edits will be overwritten the next time the Rd files are generated.

Package authors with existing Rd files will have to convert them to **inlinedocs** comments manually. However, for new adopters of inlinedocs, it is possible to mix static Rd files and **inlinedocs** in the same package. For example, the following code specifies that `file1.Rd` and `file2.Rd` are static Rd files and so should not be generated by **inlinedocs**:

```
my.parsers <- c(default.parsers, list(do.not.generate("file1","file2")))
package.skeleton.dx(parsers = my.parsers)
```

By design, **inlinedocs** is incapable of generating Rd files that document multiple objects, but package authors may write these Rd files manually using this mechanism.

More generally, the `parsers` argument to `package.skeleton.dx` should be a list of Parser Functions. In the next section, we explain how to write Parser Functions.

8.3 The inlinedocs system of extensible documentation generators

The previous section explains how to write inline documentation in R code using the standard **inlinedocs** syntax, then process it to generate Rd files using `package.skeleton.dx`. For most users of **inlinedocs** this should be sufficient for everyday use.

For users who wish to extend the syntax of **inlinedocs**, here we explain the internal organization of the **inlinedocs** package. The two central concepts are Parser Functions and Documentation Lists. Parser Functions are used to extract documentation from R code, which is then stored in a Documentation List before writing Rd files.

Documentation Lists store the structured content of Rd files

A Documentation List is a list of lists that describes all of the documentation to write to the Rd files. The elements of the outer list correspond to Rd files in the package, and the elements of the inner list correspond to tags in an Rd file. For example, consider the following code and its corresponding Documentation List.

R code

```
give.me.a.break <- function
### Create some line breaks.
(times=1,
### The number of line breaks.
 collapse=""
### String to paste in between.
 ){
  paste(rep("\n",times),
        collapse=collapse)
### Character vector of length 1.
}

give.me.five <- function
(times=1 ##<< the number of fives
 ){
  rep(5,times)
### a vector of fives
}
```

Documentation List

```
List of 2
 $ give.me.a.break:List of 5
  ..$ description   : chr "Create some line breaks."
  ..$ item{times}   : chr "The number of line breaks."
  ..$ item{collapse}: chr "String to paste in between."
  ..$ value         : chr "Character vector of length 1."
  ..$ title         : chr "give me a break"
 $ give.me.five   :List of 3
  ..$ value      : chr "a vector of fives"
  ..$ item{times}: chr "the number of fives"
  ..$ title      : chr "give me five"
```

Parser Functions examine the lines of code on the left that define the functions, and return the Documentation List of tags shown on the right. This list describes the tags in the Rd files that will be written for these functions. The names of the outer list specify the Rd file, and the names of the inner list specify the Rd tag.

To store parsed documentation, another intermediate representation that we considered instead of the Documentation List was the `"Rd"` object, as described by Murdoch and Urbanek [2009]. It is a recursive structure of lists and character strings, which is similar to the Documentation List format of **inlinedocs**. However, we chose the Documentation List format since it allows rapid development of Parser Functions which are straightforward to read, write, and modify.

Structure of a Parser Function and `forall/forfun`

The job of a Parser Function is to return a Documentation List for a package. To do this, a Parser Function requires knowledge of what is defined in the package, so the arguments in Table 8.1 are supplied by **inlinedocs**.

Argument	Description
`code`	Character vector of all lines of R code in the package.
`env`	Environment in which the lines of code are evaluated.
`objs`	List of all R objects defined in the package.
`docs`	Documentation List from previous Parser Functions.
`desc`	1-row matrix of DESCRIPTION metadata, as read by `read.dcf`.

Table 8.1: Arguments that are passed to every Parser Function.

The R code files in the package are concatenated into `code` and then parsed into `objs`, and the DESCRIPTION metadata is available as `desc`. These arguments allow complete flexibility in the construction of Parser Functions that take apart the package and extract meaningful Documentation Lists. In addition, the `docs` argument allows for checking of what previous Parser Functions have already extracted.

In principle, one could write a single monolithic Parser Function that extracts all tags for all Rd files for the package, then returns the entire Documentation List. However, in practice, this results in one unwieldly Parser Function that does many things and is hard to maintain. A simpler strategy is to write several smaller Parser Functions, each of which produces an inner Documentation List for a specific Rd file, such as the following:

```
title.from.firstline <- function (src, ...) {
  first <- src[1]
  if (grepl("#", first)) {
    list(title = gsub("[^#]*#\\s*(.*)", "\\1", first, perl = TRUE))
  } else list()
}
```

This function takes `src`, a character vector of R code lines that define a function, and looks for a comment on the first line. If there is a comment, `title.from.firstline` returns the comment as the title in an inner Documentation List. This a very simple and readable way to define a Parser Function.

Argument	Description
`o`	The R object.
`name`	The name of the object.
`src`	The source code lines that define the object.
`doc`	The inner Documentation List already constructed for this object.

Table 8.2: Arguments passed to each Parser Function, when used with `forall` or `forfun`.

But how does this Parser Function get access to the `src` argument, the source code of an individual function? We introduce the `forall` and `forfun` functions, which transform an object-specific Parser Function such as `title.from.firstline` to a Parser Function that can work on an entire package. These functions examine the `objs` and `docs` arguments, and call the object-specific Parser Function on each object in turn. The `forfun` function applies to every function in the package, whereas the `forall` function applies to every documentation object in the package.

Thus, when using a Parser Function such as `forfun(title.from.firstline)`, the additional arguments in Table 8.2 can be used in the definition of `title.from.firstline`, in addition to the arguments in Table 8.1 that are passed to every Parser Function.

This design choice of **inlinedocs** allows the development of modular Parser Functions. For example, there is one Parser Function for `###` comments, another for `##<<` comments, another for adding the `author` tag using the Author line of the DESCRIPTION file, etc. Each of these Parser Functions is relatively small and thus easy to maintain.

Extending the syntax with custom Parser Functions

The `parsers` argument to `package.skeleton.dx` specifies the list of Parser Functions used to create the Documentation List. The Parser Functions will be called in sequence, and their results will be combined to form the final Documentation List that will be used to write Rd files. Thus, the **inlinedocs** syntax can be extended by simply writing new Parser Functions. To illustrate how **inlinedocs** may be extended using this mechanism, consider this Parser Function, which extracts documentation from single-`#` comments:

```
simple <- function (src, ...) {# a simple Parser Function
  #item{src} character vector of R source code.
  noquotes <- gsub("([\"'`]).*\\1", "", src)
  comments <- grep("#", noquotes, value = TRUE)
  doc.pattern <- "[^#]*#([^ ]*) (.*)"
  tags <- gsub(doc.pattern, "\\1", comments)
  docs <- as.list( gsub(doc.pattern, "\\2", comments) )
  names(docs) <- tags
  #value all the tags with a single pound sign.
  docs[ tags != "" ]
}
```

We can then define a list of custom Parser Functions as follows:

```
simple.parsers <- list(forfun(title.from.firstline), forfun(simple))
```

These custom Parser Functions can be used to extract the following Documentation List from the definition above of `simple`:

```
List of 1
 $ simple:List of 3
  ..$ title    : chr "a simple Parser Function"
  ..$ item{src}: chr "character vector of R source code."
  ..$ value    : chr "all the tags with a single pound sign."
```

In conclusion, a new syntax for inline documentation can be quickly specified using Parser Functions, and then **inlinedocs** takes care of the details of converting the Documentation List to Rd files.

8.4 Conclusions and future work

We have presented **inlinedocs**, which is both a new syntax for inline documentation of R packages, and an extensible system for parsing this syntax and generating Rd files. It has been in development since 2009 on R-Forge [Theußl and Zeileis, 2009], has seen several releases on CRAN, and has been used to generate documentation for itself and several other R packages. In practice, we have found that **inlinedocs** significantly reduces the amount of time it takes to create a package that passes `R CMD check`. In addition, **inlinedocs** facilitates rapid package updates since the documentation is written in comments right next to the relevant code.

For quality assurance, we currently have implemented unit tests for Documentation Lists, which assure that Parser Functions work as described. We also have unit tests which ensure that the generated Rd passes `R CMD check` without errors or warnings.

A potential criticism of **inlinedocs** is that excessive inline comments may obscure the meaning of code. Indeed, this is a design choice, and can be seen as a bug, but we prefer to see it as a feature: the documentation is always near the object definition, for quick reference.

Currently, the **inlinedocs** package relies on the `srcref` attribute of a function to access its definition. For S4 classes, we use `parse` on the source files. In the future, we would like to develop Parser Functions that use this approach to extract documentation for S4 methods and reference classes, which are currently unsupported in **inlinedocs**.

For the future, we would like to make use of Rd manipulation tools such as `parse_Rd`, as described by Murdoch [2009]. For package authors who want to convert Rd files to inlinedocs comments, we may be able to use `parse_Rd` to develop a converter that takes R source code and Rd, then outputs R code with documentation in comments.

Also, it would be advantageous to have functions for converting Documentation Lists to and from `Rd` objects. For example, after converting an inner Documentation List to an `Rd` object, we could use its `print` method to write the Rd file. This could be simpler than the current system of starting from the Rd files from `package.skeleton` and then doing find and replace. Furthermore, a converter from `Rd` objects to Documentation Lists would permit unit tests for the content of the Rd generated by **inlinedocs**.

Chapter 9

Support for named capture regular expressions in R

Some material from this chapter is taken from "Fast, named capture regular expressions in R 2.14," a presentation I delivered on 16 August 2011 for the international useR conference at the University of Warwick, England.

Chapter summary

Regular expressions are powerful tools for text processing, and are widely used by statisticians for pre-processing data files before analysis. For example, the regular expression `[0-9]` will match any single digit, and the regular expression `[0-9]+` will match one or more digits. These can be used to extract numbers from non-standard data files such as:

```
chromStart=400000, chromEnd=800000, annotation=1
chromStart=1200000, chromEnd=2400000, annotation=0
...
```

This chapter discusses the use of regular expressions in the R programming language [R Development Core Team, 2012]. I will focus on how I implemented the following features:

- **Capturing subpattern locations**. This allows extracting several different substrings using a single regular expression.
- **Capturing subpattern names**. This allows extracting substrings using names defined in the regular expression.

As a result of this work, these features are available from the `regexpr` and `gregexpr` functions, in every copy of R starting with version 2.14.

9.1 Introduction and related work

Let us consider an example of how to use regular expressions for text processing, taken from earlier in this thesis. The lines below show the header of a data file that I received from a collaborator:

```
> print(header)
[1] "DELETION 1p    0=pas de deletion  1=deletion      9=NI"
[2] " GAIN 2p            0=pas de gain  1=gain  3=amplicon MYCN 9"
[3] "DELETION  3p     0=pas de deletion  1=deletion        9=NI"
[4] "DELETION  4p     0=pas de deletion  1=deletion       9=NI"
[5] "DELETION  11q     0=pas de deletion  1=deletion    9=NI"
[6] " GAIN 17q            0=pas de gain 17q  1=gain  17q      9=N"
```

The header indicates the chromosome arm as 1p, 2p, etc. I needed to convert this chromosome arm information to genome coordinates to define the regions of the breakpoint annotations shown in Table 3.2. The first step in this conversion is extracting the chromosome arm from the header. Ideally, parsing the header would yield the information in Table 9.1.

In fact, this can be accomplished programmatically in many ways. In R-2.13, we can try to use the `regexpr` function:

```
> regexpr("[0-9]+[pq]",header,perl=TRUE)
[1] 10  7 11 11 11  7
attr(,"match.length")
[1] 2 2 2 2 3 3
```

However, `regexpr` returns only the locations where the entire pattern matches. Since it does not tell us where the chromosome ends and the arm begins, it does not really help us to create Table 9.1.

chr	arm
1	p
2	p
3	p
4	p
11	q
17	q

Table 9.1: Chromosome arm data that we would like to extract from the header of a data file.

Other text processing functions such as `sub` can be used for extracting matched subpatterns, but that requires some repetition in the code:

```
> pattern2 <- ".* ([0-9]+)([pq]) .*"
> cbind(chr=sub(pattern2,"\\1",header),
+       arm=sub(pattern2,"\\2",header))
     chr  arm
[1,] "1"  "p"
[2,] "2"  "p"
[3,] "3"  "p"
[4,] "4"  "p"
[5,] "11" "q"
[6,] "17" "q"
```

Using this method, the group names can be defined later in R code, but are separated from the regular expression pattern.

Another method is to use the `stringr::str_match` function [Wickham, 2010], which extracts capture groups:

```
> library(stringr)
> str_match(header,"([0-9]+)([pq])")
     [,1]  [,2] [,3]
[1,] "1p"  "1"  "p"
[2,] "2p"  "2"  "p"
[3,] "3p"  "3"  "p"
[4,] "4p"  "4"  "p"
[5,] "11q" "11" "q"
[6,] "17q" "17" "q"
```

However, it has two drawbacks:

- Group names can be defined later in R code, but are separated from the regular expression pattern.
- In R-2.13, the `str_match` function extracts capture groups using R code. So it usually runs much slower than the C implementation we propose in the next section.

In R-2.14, a named capture regular expression can be used:

$$\overset{\text{name 1}}{\overbrace{\texttt{(?<chr>}}}\underbrace{\texttt{[0-9]+}}_{\text{subpattern 1}}\texttt{)(?<}\overset{\text{name 2}}{\overbrace{\texttt{arm}}}\texttt{>}\underbrace{\texttt{[pq]}}_{\text{subpattern 2}}\texttt{)} \tag{9.1}$$

This regular expression consists of 2 groups, delimited using parentheses. Inside each group is its `?<name>` followed by its pattern. Regular expression 9.1 defines everything we need to extract the chromosome arm information and create Table 9.1.

First, we can use the following code to get the names and match locations of each subpattern:

```
> regexpr("(?<chr>[0-9]+)(?<arm>[pq])",header,perl=TRUE)
[1] 10  7 11 11 11  7
attr(,"match.length")
[1] 2 2 2 2 3 3
attr(,"useBytes")
[1] TRUE
attr(,"capture.start")
     chr arm
[1,]  10  11
[2,]   7   8
[3,]  11  12
[4,]  11  12
[5,]  11  13
[6,]   7   9
attr(,"capture.length")
     chr arm
[1,]   1   1
[2,]   1   1
[3,]   1   1
[4,]   1   1
[5,]   2   1
[6,]   2   1
attr(,"capture.names")
[1] "chr" "arm"
```

In particular, the start and the length of each matched subpattern is returned in the `capture.start` and `capture.length` attributes. Furthermore, the group names are returned in the `capture.names` attribute.

Then, we can build a function `str_match_perl` that parses the output of `regexpr` and returns a matrix of matched substrings:

```
str_match_perl <- function(string,pattern){
  parsed <- regexpr(pattern,string,perl=TRUE)
  captured.text <- substr(string,parsed,parsed+attr(parsed,"match.length")-1)
  captured.text[captured.text==""] <- NA
  captured.groups <- do.call(rbind,lapply(seq_along(string),function(i){
    st <- attr(parsed,"capture.start")[i,]
    if(is.na(parsed[i]) || parsed[i]==-1)return(rep(NA,length(st)))
    substring(string[i],st,st+attr(parsed,"capture.length")[i,]-1)
  }))
  result <- cbind(captured.text,captured.groups)
  colnames(result) <- c("",attr(parsed,"capture.names"))
  result
}
```

So it is possible to create Table 9.1 in 1 line of R code using named capture regular expressions:

```
> str_match_perl(header,"(?<chr>[0-9]+)(?<arm>[pq])")
           chr  arm
[1,] "1p"  "1"  "p"
[2,] "2p"  "2"  "p"
[3,] "3p"  "3"  "p"
[4,] "4p"  "4"  "p"
[5,] "11q" "11" "q"
[6,] "17q" "17" "q"
```

The definitive reference on regular expressions is [Friedl, 2006], which describes the concept of regular expressions and how to make the best use of them.

Named capture regular expressions were first implemented in the Python programming language. Later, they were implemented in the Perl programming language, and in the Perl-Compatible Regular Expressions (PCRE) library used by R [Hazel, 2012]. Although PCRE has always supported named capture regular expressions, R has only been able to starting with version 2.14, as a result of this work.

9.2 Implementation details

In this section, I will discuss the details of the PCRE library that allowed implementation of named capture regular expressions in R.

In PCRE terminology, the **pattern** is the regular expression and the **subject** is the string where it will look for a match.

To use named capture regular expressions, we must first use the `pcre_fullinfo` C function to obtain some information about the capture groups. The `pcre_fullinfo` function takes a compiled pattern `re_pcre`, a `pointer`, and a data type `CONSTANT`, and stores some information at that `pointer`:

```
pcre_fullinfo(re_pcre, NULL, CONSTANT, pointer)
```

In particular, we need the following constants:

- `PCRE_INFO_CAPTURECOUNT` the number of capture groups.
- `PCRE_INFO_NAMETABLE` pointer to a table of characters that stores the group names.
- `PCRE_INFO_NAMECOUNT` the number of named groups.
- `PCRE_INFO_NAMEENTRYSIZE` the size of each entry in the group name table.

To explain the structure of the group name table, we quote the example from the PCRE manual [Hazel, 2012, section Information about a pattern]:

```
As a simple example of the name/number table, consider the following
pattern after compilation by the 8-bit library (assume PCRE_EXTENDED is
set, so white space - including newlines - is ignored):

 (?<date> (?<year>(\d\d)?\d\d) -
 (?<month>\d\d) - (?<day>\d\d) )

There are four named subpatterns, so the table has four entries, and
each entry in the table is eight bytes long. The table is as follows,
with non-printing bytes shows in hexadecimal, and undefined bytes shown
as ??:

 00 01 d  a  t  e  00 ??
 00 05 d  a  y  00 ?? ??
 00 04 m  o  n  t  h  00
 00 02 y  e  a  r  00 ??
```

So to access the group names in R, we simply copy data from the group name table to the `capture.names` attribute of the result of `regexpr`.

When PCRE finds a match, it stores that match in an object called an output vector or ovector. If the pattern has n capturing groups, the ovector should be of size $3(n+1)$. Continuing the example from the previous page, the date pattern has $n = 5$ capturing groups so the ovector should be of size 18. As shown in Figure 9.1, the first 2/3 contains the start and end locations of matches, and the last 1/3 is a workspace which is required but can be ignored.

So to access the text matched by subpatterns, we just need to copy the values in the ovector to an R integer vector. Two caveats to keep in mind:

- In C the ovector start is a 0-indexed byte offset, and we convert it to a 1-indexed character offset in R.
- In C the ovector end is the 0-indexed byte after the last matching byte, and we convert this to a match length in characters in R.

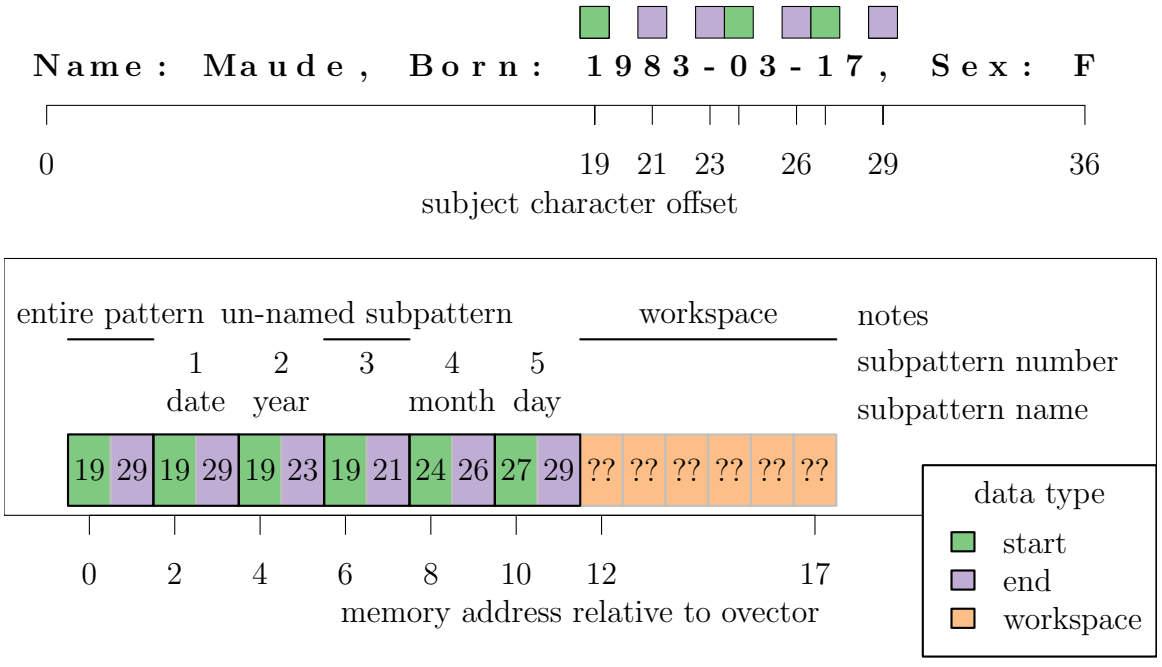

Figure 9.1: **Top**: subject string with match start and end locations indicated using colored squares. **Bottom**: memory layout of the PCRE output vector.

9.3 Application: extracting data from HTML

In this section, we will show how named capture regular expressions can be used to extract data from web pages.

Web pages are written in Hyper-Text Markup Language (HTML), which can be rendered using a web browser. In Figure 9.2, we show an R-Forge web page, and some of its corresponding HTML source code. The object of this section will be to extract data from this HTML for analysis in R.

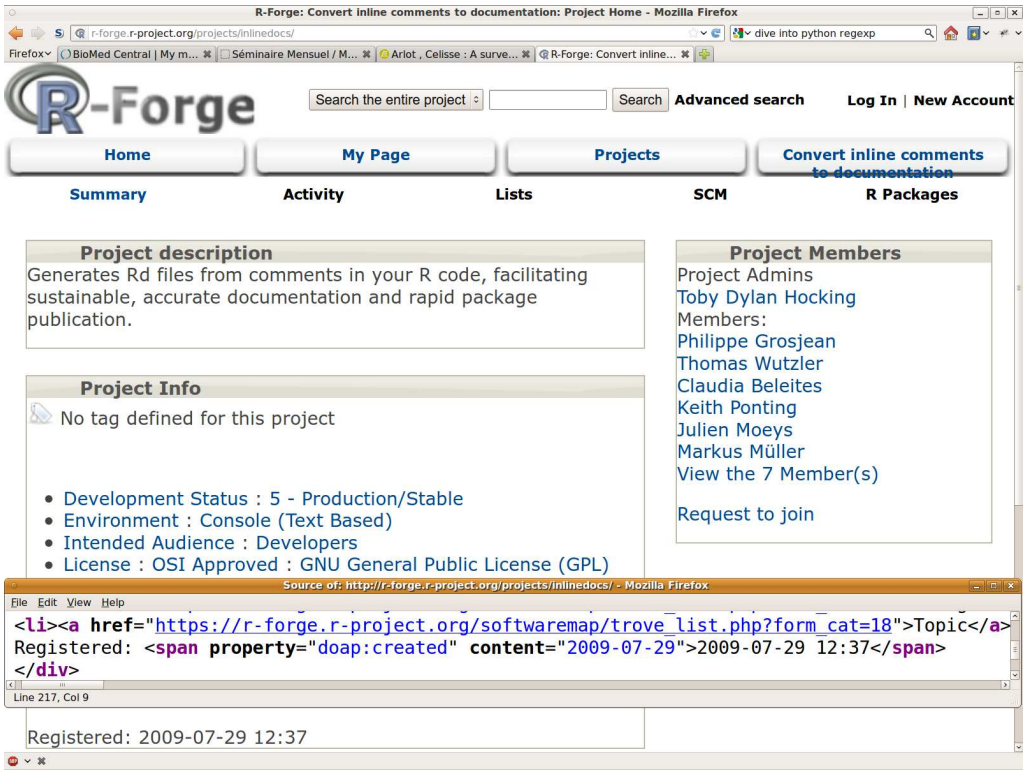

Figure 9.2: **Top**: a web page with information about a project on R-Forge. **Bottom**: some of the HTML source code of this web page. We would like to extract the project registration date `2009-07-29` for analysis.

Using the functions discussed in the previous sections, we can extract the project registration date from several web pages using

```
> getHTML <- function(proj,tmp="http://r-forge.r-project.org/projects/%s/"){
+   u <- sprintf(tmp,proj)
+   paste(readLines(url(u)),collapse="\n")
+ }
> html <- sapply(c("inlinedocs","directlabels","clusterpath"),getHTML)
> pat <- paste("(?<year>(\\d\\d)?\\d\\d)",
+              "-",
+              "(?<month>\\d\\d)",
+              "-",
+              "(?<day>\\d\\d)",
+              sep="")
> str_match_perl(html,pat)
                           year        month day
inlinedocs   "2009-07-29" "2009" "20" "07"  "29"
directlabels "2009-07-17" "2009" "20" "07"  "17"
clusterpath  "2011-05-09" "2011" "20" "05"  "09"
```

And using that approach on all the R-Forge project web pages, we can plot the growth of R-Forge in Figure 9.3. In conclusion, regular expressions are useful for parsing numeric data sets out of text files such as HTML.

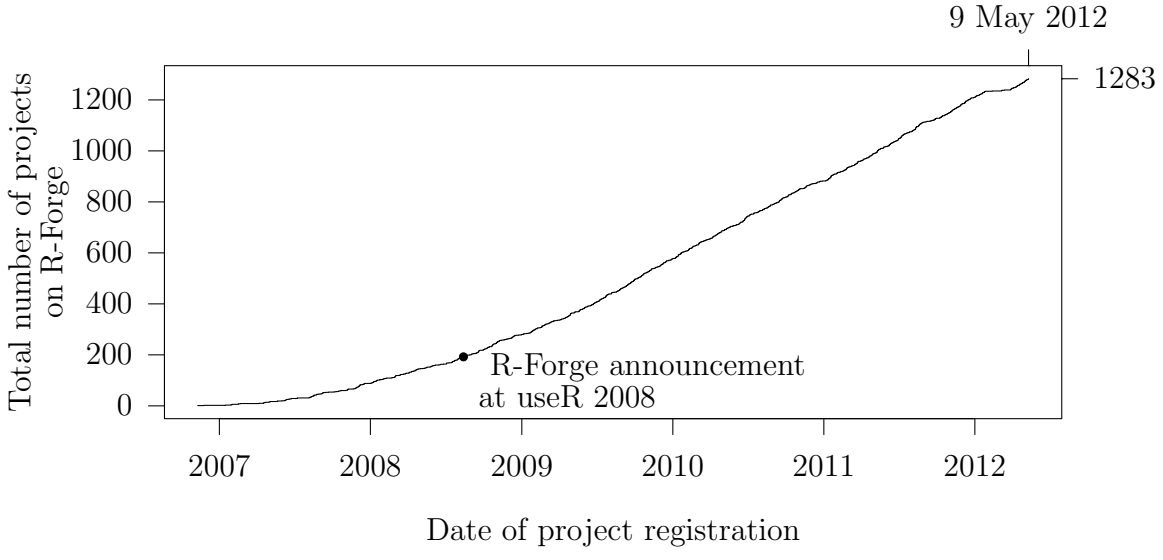

Figure 9.3: R-Forge project web pages were downloaded and parsed using regular expressions, then analyzed to plot the number of registered projects over time.

9.4 Conclusion

In this chapter, we described how regular expressions may be used for preprocessing data files before statistical analysis. Then, we discussed how named capture regular expressions can be implemented in R by exploiting the PCRE library. Finally, we showed an application to extracting data from web pages.

Starting from R-2.14, the code that implements named capture regular expressions is included in the `regexpr` and `gregexpr` functions. Table 9.2 compares the regular expression functions in R-2.13 and R-2.14. It is clear that the regular expression functionality in R has been augmented as a result of this work.

	R 2.13 `regexpr`	R 2.13 `str_match`	R 2.14 `regexpr`
whole match	✓	✓	✓
capture		✓	✓
fast C code	✓		✓
named capture			✓

Table 9.2: Summary of contributions for text processing with named capture regular expressions.

Bibliography

H. Akaike. Information theory as an extension of the maximum likelihood principle. In B. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*, pages 267–281. Akademiai Kiado, Budapest, 1973.

B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, New York, fourth edition, 2002.

S. Arlot. V-fold cross-validation improved: V-fold penalization. *Arxiv preprint arXiv:0802.0566*, 2008.

S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

S. Arlot and P. Massart. Data-driven calibration of penalties for least-squares regression. *J. Mach. Learn. Res.*, 10:245–279, June 2009. ISSN 1532-4435. `http://dl.acm.org/citation.cfm?id=1577069.1577079`.

F. Bach and Z. Harchoui. DIFFRAC: a discriminative and flexible framework for clustering. In *Adv. NIPS*, 2008.

F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

Y. Baraud, C. Giraud, and S. Huet. Gaussian model selection with unknown variance. *Ann. Statist.*, 37(2):630–672, 2009.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, 2(1):183–202, 2009.

E. Ben-Yaacov and Y. C. Eldar. A Fast and Flexible Method for the Segmentation of aCGH Data. *Bioinformatics*, 24(16):i139–i145, September 2008.

H. Bengtsson. The **R.oo** package - object-oriented programming with references using standard R code. In K. Hornik, F. Leisch, and A. Zeileis, editors, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, Vienna, Austria, March 2003.

H. Bengtsson. **Aroma** project developers' corner, 2010. URL `http://www.aroma-project.org/developers`.

L. Birgé and P. Massart. Minimal penalties for gaussian model selection. *Probability Th. and Related Fields*, 138:33–73, 2007.

M. Bostock, V. Oglevetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12): 2301–2309, December 2011.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, The Edinburgh Building, Cambridge, CB2 8RU, UK, 2004.

E. J. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Trans. Inform. Theory*, 56(5):2053–2080, 2009.

X. Chen, S. Kim, Q. Lin, J. G. Carbonell, and E. P. Xing. Graph-structured multi-task regression and an efficient optimization method for general fused lasso, 2010. arXiv:1005.3579.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, Cambridge, Massachusetts, second edition, 1990.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 26. MIT Press, 2001.

P. Danenberg. **Roxygen** vignette, 2009. http://cran.r-project.org/web/packages/roxygen/vignettes/roxygen.pdf.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. http://www.R-project.org.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of statistics*, 32(2):40–99, 2004.

H. Fiegler, R. Redon, D. Andrews, C. Scott, R. Andrews, C. Carder, R. Clark, O. Dovey, P. Ellis, L. Feuk, L. French, P. Hunt, D. Kalaitzopoulos, J. Larkin, L. Montgomery, G. H. Perry, B. W. Plumb, K. Porter, R. E. Rigby, D. Rigler, A. Valsesia, C. Langford, S. J. Humphray, S. W. Scherer, C. Lee, M. E. Hurles, and N. P. Carter. Accurate and reliable high-throughput detection of copy number variation in the human genome. *Genome Res.*, 16(12):1566–1574, Dec. 2006. doi: 10.1101/gr.5630906. URL `http://dx.doi.org/10.1101/gr.5630906`.

A. Fischer. On the number of groups in clustering. *Statistics and Probability Letters*, 81:1771–1781, 2011.

M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95—110, 1956.

J. E. Friedl. *Mastering Regular Expressions*. O'Reilly, 1005 Gravenstein Highway North, Sebastopol, CA 95472, third edition, 2006.

J. Friedman, T. Hastie, H. Hoefling, and R. Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):30–32, 2007.

S. Fujishige, T. Hayashi, and S. Isotani. The minimum-norm-point algorithm applied to submodular function minimization and linear programming, 2006. RIMS preprint No 1571. Kyoto University.

R. C. Gentleman, V. J. Carey, D. M. Bates, and others. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. `http://genomebiology.com/2004/5/10/R80`.

D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1983.

M. Grant, S. Boyd., and Y. Ye. *Global Optimization: From Theory to Implementation*, chapter Disciplined convex programming. Springer, 2006.

P. Hall, J. W. Kay, and D. Titterinton. Asymptotically optimal difference-based estimation of variance in nonparametric regression. *Biometrika*, 77 (3):521–528, January 1990.

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, Springer Science+Business Media, LLC, 233 Spring Street, New York NY 10013, USA, second edition, 2009.

P. Hazel. PCRE - Perl-compatible regular expressions (man page), 2012. http://pcre.org/pcre.txt.

T. D. Hocking, A. Joulin, F. Bach, and J.-P. Vert. Clusterpath: an algorithm for clustering using convex fusion penalties. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 745–752, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

T. D. Hocking, G. Schleiermacher, I. Janoueix-Lerosey, O. Delattre, F. Bach, and J.-P. Vert. Learning smoothing models using breakpoint annotations. Technical report `http://hal.inria.fr/hal-00663790`, 2012.

T. D. Hocking, T. Wutzler, K. Ponting, and P. Grosjean. Sustainable, Extensible Documentation Generation using **inlinedocs**. *Journal of Statistical Software*, in press.

H. Hoefling. A path algorithm for the Fused Lasso Signal Approximator. arXiv:0910.0526, 2009.

L. Hubert and P. Arabie. Comparing partitions. *J. Classification*, 2:193–218, 1985.

P. Hupé, N. Stransky, J.-P. Thiery, F. Radvanyi, and E. Barillot. Analysis of array CGH data: from signal ratio to gain and loss of DNA regions. *Bioinformatics*, 20(18):3413–3422, 2004.

IHGSC. Finishing the euchromatic sequence of the human genome. *Nature*, 431:931–945, October 2004. IHGSC: International Human Genome Sequencing Consortium.

I. Janoueix-Lerosey, D. Lequin, L. Brugières, A. Ribeiro, L. de Pontual, V. Combaret, V. Raynal, A. Puisieux, G. Schleiermacher, G. Pierron, D. Valteau-Couanet, T. Frebourg, J. Michon, S. Lyonnet, J. Amiel5, and O. Delattre. Somatic and germline activating mutations of the alk kinase receptor in neuroblastoma. *Nature*, 344:967–970, October 2008.

I. Janoueix-Lerosey, G. Schleiermacher, E. Michels, V. Mosseri, A. Ribeiro, D. Lequin, J. Vermeulen, J. Couturier, M. Peuchmaur, A. Valent, D. Plantaz, H. Rubie, D. Valteau-Couanet, C. Thomas, V. Combaret, R. Rousseau, A. Eggert, J. Michon, F. Speleman, and O. Delattre. Overall genomic pattern is a predictor of outcome in neuroblastoma. *Journal of Clinical Oncology*, 27(7):1026–1033, 2009. doi: 10.1200/JCO.2008.16.0630. `http://jco.ascopubs.org/content/27/7/1026.abstract`.

T. R. Jones, A. E. Carpenter, M. R. Lamprecht, J. Moffat, S. J. Silver, J. K. Grenier, A. B. Castoreno, U. S. Eggert, D. E. Root, P. Golland, and D. M. Sabatini. Scoring diverse cellular morphologies in image-based screens with iterative feedback and machine learning. *Proceedings of the National Academy of Sciences*, 106(6):1826–1831, 2009. doi: 10.1073/pnas.0808843106. `http://www.pnas.org/content/106/6/1826.abstract`.

W. Kent, C. Sugnet, T. Furey, K. Roskin, T. Pringle, A. Zahler, and D. Haussler. The human genome browser at UCSC. *Genome Research*, 12(6):996–1006, June 2002.

B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Inc., 1988.

R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. arXiv:1101.1438, 2011.

A. Krause and C. Guestrin. Beyond convexity: Submodularity in machine learning. In *IJCAI*, 2009.

P. La Rosa, E. Viara, P. Hupé, G. Pierron, S. Liva, P. Neuvial, I. Brito, S. Lair, N. Servant, N. Robine, E. Manié, C. Brennetot, I. Janoueix-Lerosey, V. Raynal, N. Gruel, C. Rouveirol, N. Stransky, M.-H. Stern, O. Delattre, A. Aurias, F. Radvanyi, and E. Barillot. VAMP: Visualization and analysis of array-CGH, transcriptome and other molecular profiles. *Bioinformatics*, 22(17):2066–2073, 2006. doi: 10.1093/bioinformatics/btl359. `http://bioinformatics.oxfordjournals.org/content/22/17/2066.abstract`.

L. Lamport. ***LaTeX****: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 1986.

M. Lavielle. Using penalized contrasts for the change-point problem. *Signal Processing*, 85:1501–1510, 2005.

E. Lebarbier. Detecting multiple change-points in the mean of gaussian process by model selection. *Signal Processing*, 85:717–736, 2005.

C.-B. Lee. Estimating the number of change points in a sequence of independent normal random variables. *Statist. Proba. Lett.*, 25(3):241–8, 1995.

F. Leisch. **Sweave**, part II: Package vignettes. *R News*, 3(2):21–24, October 2003. http://CRAN.R-project.org/doc/Rnews/.

F. Lindsten, H. Ohlsson, and L. Ljung. Clustering using sum-of-norms regularization; with application to particle filter output computation. Technical Report LiTH-ISY-R-2993, Department of Electrical Engineering, Linköping University, Feb. 2011.

J. M. Maris. Recent advances in neuroblastoma. *New England Journal of Medicine*, 362(23):2202–2211, 2010. doi: 10.1056/NEJMra0804577. URL `http://www.nejm.org/doi/full/10.1056/NEJMra0804577`.

J. Mattingley and S. Boyd. CVXMOD: Convex optimization software in Python (web page and software), July 2008. `http://cvxmod.net/`.

D. Murdoch. *Parsing Rd files*, 2009. http://developer.r-project.org/parseRd.pdf.

D. Murdoch and S. Urbanek. The New R Help System. *The R Journal*, 1(2):60–65, December 2009. http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Murdoch+Urbanek.pdf.

P. Murrell. *Introduction to Data Technologies*. CRC computer science and data analysis series. Chapman & Hall, Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742, 2009.

P. Murrell. *R Graphics*. The R Series. CRC Press, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742, second edition, 2011.

M. T. Nakao, A. Neumaier, S. M. Rump, S. P. Shary, and P. van Hentenryck. Standardized notation in interval analysis. `http://www.mat.univie.ac.at/~neum/papers.html`, 2010.

P. Neuvial, H. Bengtsson, and T. P. Speed. Statistical analysis of single nucleotide polymorphism microarrays in cancer studies. Technical report `http://hal.inria.fr/hal-00497273`, 2010.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Adv. NIPS*, 2001.

K. Pelckmans, J. de Brabanter, and J. Suykens. Convex clustering shrinkage. In *Statistics and Optimization of Clustering Workshop (PASCAL)*, London, UK, July 2005.

F. Picard, S. Robin, M. Lavielle, C. Vaisse, and J.-J. Daudin. A statistical approach for array CGH data analysis. *BMC Bioinformatics*, 6(27), 2005.

M. Pilgrim. *Dive into Python*. Apress, July 2004.

D. Pinkel, R. Segraves, D. Sudar, S. Clark, I. Poole, D. Kowbel, C. Collins, W.-L. Kuo, C. Chen, Y. Zhai, S. H. Dairkee, B.-m. Ljung, J. W. Gray, and D. G. Albertson. High resolution analysis of DNA copy number variation using comparative genomic hybridization to microarrays. *Nature Genetics*, 20(2):207–211, Oct. 1998. ISSN 1061-4036. doi: 10.1038/2524. http://dx.doi.org/10.1038/2524.

R. Pique-Regi, J. Monso-Varona, A. Ortega, R. C. Seeger, T. J. Triche, and S. Asgharzadeh. Sparse representation and Bayesian detection of genome copy number alterations from microarray data. *Bioinformatics*, 24(3): 309–318, 2008.

G. Rigaill. Pruned dynamic programming for optimal multiple change-point detection. arXiv:1004.0887, 2010.

A. Ritz, P. Paris, M. Ittmann, C. Collins, and B. Raphael. Detection of recurrent rearrangement breakpoints from copy number data. *BMC Bioinformatics*, 12(1):114, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-114. `http://www.biomedcentral.com/1471-2105/12/114`.

S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *Annals of Statistics*, 35(3):1012–1030, 2007.

A. J. Rossini, R. M. Heiberger, R. A. Sparapani, M. Maechler, and K. Hornik. **Emacs Speaks Statistics**: A multiplatform, multipackage development environment for statistical analysis. *Journal of Computational and Graphical Statistics*, 13(1):247–261, 2004.

B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1–3):157–173, May 2008.

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL `http://lmdvr.r-forge.r-project.org`. ISBN 978-0-387-75968-5.

G. Schleiermacher, I. Janoueix-Lerosey, A. Ribeiro, J. Klijanienko, J. Couturier, G. Pierron, V. Mosseri, A. Valent, N. Auger, D. Plantaz, H. Rubie, D. Valteau-Couanet, F. Bourdeaut, V. Combaret, C. Bergeron, J. Michon, and O. Delattre. Accumulation of segmental alterations

determines progression in neuroblastoma. *Journal of Clinical Oncology*, 28(19):3122–3130, 2010. doi: 10.1200/JCO.2009.26.7955. `http://jco.ascopubs.org/cgi/content/abstract/28/19/3122`.

M. Schwab, H. Varmus, J. Bishop, K. Grzeschik, S. Naylor, A. Sakaguchi, G. Brodeur, and J. Trent. Chromosome localization in normal human cells and neuroblastomas of a gene related to c-myc. *Nature*, 308(5956): 288–291, March 1984.

G. Schwarz. Estimating the dimension of a model. *Ann. Statist.*, 6(2): 461–464, 1978.

S. P. Shah, X. Xuan, R. J. DeLeeuw, M. Khojasteh, W. L. Lam, R. Ng, and K. P. Murphy. Integrating copy number polymorphisms into array CGH analysis using a robust HMM. *Bioinformatics*, 22(14):431–439, 2006.

X. Shen and H.-C. Huang. Grouping pursuit through a regularization solution surface. *Journal of the American Statistical Association*, 105(490): 727–739, 2010.

B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, third edition, 1997.

S. Theußl and A. Zeileis. Collaborative Software Development Using R-Forge. *The R Journal*, 1(1):9–14, May 2009. `http://journal.r-project.org/2009-1/RJournal_2009-1_Theussl+Zeileis.pdf`.

R. Tibshirani. Regression Shrinkage and Selection Via the Lasso. *J. R. Statist. Soc. B.*, 58(1):267–288, 1996.

R. Tibshirani and M. Saunders. Sparsity and smoothness via the fused lasso. *J. R. Statist. Soc. B.*, 67:9–08, 2005.

R. Tibshirani and P. Wang. Spatial smoothing and hot spot detection for CGH data using the fused lasso. *Biostatistics*, 2007.

R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *J. R. Statist. Soc. B*, 63:41–23, 2001.

D. Trochet, F. Bourdeaut, I. Janoueix-Lerosey, A. Deville, L. de Pontual, G. Schleiermacher, C. Coze, N. Philip, T. Frébourg, A. Munnich, S. Lyonnet, O. Delattre, and J. Amiel. Germline mutations of the paired–like homeobox 2b (phox2b) gene in neuroblastoma. *Am. J. Hum. Genet.*, 74: 761–764, 2004.

E. Tufte. *The Visual Display of Quantitative Information*. Graphics Press LLC, Post office box 430, Cheshire, Connecticut 06410, second edition, 2001.

E. Tufte. *Beautiful Evidence*. Graphics Press LLC, Post office box 430, Cheshire, Connecticut 06410, 2006.

B. A. Turlach and A. Weingessel. *quadprog: Functions to solve Quadratic Programming Problems.*, 2011. URL `http://CRAN.R-project.org/package=quadprog`. R package version 1.5-4. S original by Berwin A. Turlach and R port by Andreas Weingessel <Andreas.Weingessel@ci.tuwien.ac.at>.

V. Vapnik, S. Golowich, and A. J. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9 (NIPS)*, pages 281–287, 1997.

E. S. Venkatraman and A. B. Olshen. A faster circular binary segmentation algorithm for the analysis of array CGH data. *Bioinformatics*, 23(6):657–663, Mar. 2007. ISSN 1367-4811. doi: 10.1093/bioinformatics/btl646. `http://dx.doi.org/10.1093/bioinformatics/btl646`.

J.-P. Vert and K. Bleakley. Fast detection of multiple change-points shared by many signals using group LARS. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Cullota, editors, *Advances in Neural Information Processing Systems 23 (NIPS)*, pages 2343–2351, 2010.

R. A. Weinberg. *The Biology of Cancer*. Garland Science, first edition, June 2006.

H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN 978-0-387-98140-6. http://had.co.nz/ggplot2/book.

H. Wickham. stringr: modern, consistent string processing. *R Journal*, 2 (2):38–40, 2010.

Wikipedia. Comparison of documentation generators, 2012a. http://en.wikipedia.org/w/index.php?title=Comparison_of_documentation_generators&oldid=507629530.

Wikipedia. Automatic label placement, 2012b. http://en.wikipedia.org/w/index.php?title=Automatic_label_placement&oldid=506419827.

H. Willenbrock and J. Fridlyand. A comparison study: applying segmentation to array CGH data for downstream analysis. *Bioinformatics*, 21 (22):4084–4091, 2005.

L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In *Adv. NIPS*, 2004.

Y.-C. Yao. Estimating the number of change-points via Schwarz' criterion. *Statistics & Probability Letters*, 6(3):181–189, February 1988. URL `http://ideas.repec.org/a/eee/stapro/v6y1988i3p181-189.html`.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society*, 68(B):4–7, 2006.

N. R. Zhang and D. O. Siegmund. A Modified Bayes Information Criterion with Applications to the Analysis of Comparative Genomic Hybridization Data. *Biometrics*, 63:22–32, 2007.

Z. Zhang, K. Lange, R. Ophoff, and C. Sabatti. Reconstructing DNA copy number by penalized estimation and imputation. *The Annals of Applied Statistics*, 4:1749–1773, 2010.

P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *Ann. Stat.*, 37(6A):3468–3497, 2009.