12-2016

A framework for the statistical analysis of mass spectrometry imaging experiments

Kyle Bemis
Purdue University

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Kyle Dwayne Bemis

Entitled
A FRAMEWORK FOR THE STATISTICAL ANALYSIS OF MASS SPECTROMETRY IMAGING EXPERIMENTS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Olga Vitek
Co-chair

Hyonho Chun
Co-chair

Hao Zhang

R. Graham Cooks

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Hyonho Chun

Approved by: Hao Zhang August 22, 2016

Head of the Departmental Graduate Program Date

A FRAMEWORK FOR THE STATISTICAL ANALYSIS OF MASS SPECTROMETRY IMAGING EXPERIMENTS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Kyle Bemis

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

West Lafayette, Indiana

For my mother and father. For my brother, my sister-in-law, and my niece.
For my Native family and my trans sisters. May you have the strength to make your dreams come true.

ACKNOWLEDGMENTS

I would like to thank my PhD committee for their generosity, for giving me so much of their valuable time and expert advice. I give a special thanks to my major professor Dr. Olga Vitek, who has been incredible in every way a major professor can be. I thank her for her patience and guidance in all aspects of academia, research, and life. I couldn't ask for a better PhD advisor. Thank you, Dr. Olga Vitek, Dr. Hyonho Chun, Dr. Hao Zhang, Dr. Bowei Xi, and Dr. R. Graham Cooks.

I thank the collaborators with whom I've had the privilege to work at Purdue and elsewhere. Thank you, Livia Eberlin, Christina Ferreira, and everyone else from Dr. R. Graham Cook's lab. You first introduced me to imaging mass spectrometry, helped teach me when I was a clueless statistician, and helped me in so many other ways with this research. Thank you, Stephanie Van de Ven, Uma Kota, Parag Mallick, Mark Stolowitz, and everyone else from the Canary Center at Stanford. You re-invigorated my interest in research in the wake of terrifying qualifying exams, made much of this work possible, and helped me on the road to my first publication. A statistician is nothing without her collaborators, and I greatly valued working with all of you. This dissertation would not exist without you.

I thank Dr. Rebecca W. Doerge and Dr. Mary Ellen Bock for their friendship and guidance, and for making Purdue Statistics the welcoming and diverse place that it is today. I am eternally grateful to have such inspiring and strong role models in Dr. Rebbeca W. Doerge, Dr. Mary Ellen Bock, and Dr. Olga Vitek.

I thank all of Dr. Olga Vitek's research group for their help, support, and friendship. Especially April Harry, who has been the most amazing research partner in the world, and has contributed countless hours to the work that made this dissertation possible. Thank you for supporting me through many crises I never thought I'd face. I thank all of the other Purdue Statistics students who have been with me from the beginning. Thank you Meena Choi, Faye Zheng, Kelly-Ann Dixon Hamil, and many others, for giving me a great time at Purdue Statistics.

I thank everyone from the Native American Educational and Cultural Center at Purdue for giving me a family away from home, and supporting me through many tough times. Thank you, Felica Ahasteen-Bryant and Debra Swihart for giving us a place to call home on campus. Thank you Darryl Reano, Wai Allen, Janelle Cronin, Kyle Dahlin, Jordan Oshiro, Rylan Chong, Mike King, Michelle Yatchmeneff, Patrick Austin Freeland, Studie Red Corn, RaeLynn Butler, Emily Head, and the rest of my Native family at Purdue. We had great times together, and you will all always have a special place in my heart. Elahkwa.

Thank you, Ken Ridgeway, Kevin Gibson, and everyone else from the Sloan Indigenous Graduate Partnership program at Purdue and elsewhere, especially those from the Montana programs, who all welcomed me with open arms. Your support for me and the rest of my Native family at Purdue throughout graduate school has been priceless. I would like to give a special thanks to Ken Ridgeway for offering so much wisdom throughout many great discussions at Purdue, in Alaska and Yellowstone, and on the road. Every field trip needs a Native geologist.

Finally, I thank my family. I thank my brother Kirk, my sister-in-law Esther, and my niece Raina for their acceptance and support of me in all aspects of life. I thank my father for giving me the wisdom and inspiration to begin this work. I thank my mother for giving me the strength to perservere and complete it.

Thank you, all. Elahkwa.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

DESI	desorption electrospray ionization
MALDI	matrix-assisted laser desorption/ionization
MS	mass spectrometry
m/z	mass-to-charge ratio
O-PLS-DA	orthogonal projection to latent structures discriminant analysis
PLS-DA	projection to latent structures distriminant analysis
RCC	renal cell carcinoma
SA	spatially-aware
SASA	spatially-aware structurally-adaptive
TOF	time-of-flight

ABSTRACT

Bemis, Kyle PhD, Purdue University, December 2016. A Framework for the Statistical Analysis of Mass Spectrometry Imaging Experiments. Major Professor: Olga Vitek.

Mass spectrometry (MS) imaging is a powerful investigation technique for a wide range of biological applications such as molecular histology of tissue, whole body sections, and bacterial films , and biomedical applications such as cancer diagnosis. MS imaging visualizes the spatial distribution of molecular ions in a sample by repeatedly collecting mass spectra across its surface, resulting in complex, high-dimensional imaging datasets. Two of the primary goals of statistical analysis of MS imaging experiments are classification (for supervised experiments), i.e. assigning pixels to pre-defined classes based on their spectral profiles, and segmentation (for unsupervised experiments), i.e. assigning pixels to newly discovered segments with relatively homogenous and distinct spectral profiles. To accomplish these goals, this research provides both statistical methods and statistical computing tools. First, we propose a novel spatial shrunken centroids framework for performing classification and segmentation of MS imaging experiments with feature selection. Spatial shrunken centroids combines spatial smoothing with statistical regularization in a model-based framework appropriate for both supervised and unsupervised settings. Second, we provide *Cardinal* , a free and open-source R package for processing, visualization, and statistical analysis of MS imaging experiments. *Cardinal* is the first R package designed specifically for MS imaging, and the first software for MS imaging that focuses specifically on experiments and statistical analysis. In addition to providing tools for statistical analysis, it also provides infrastructure to enable other statisticians to more easily develop new methods for MS imaging experiments. Lastly, to enable scalability of *Cardinal* to larger-than-memory datasets, we provide *matter* , a free and open-source R package for statistical computing with structured datasets-on-disk, such as MS imaging data files. Together, spatial shrunken centroids, *Cardinal* , and *matter* aim to allow scalable statistical analysis for high-resolution, high-throughput MS imaging experiments.

1. PROBLEM STATEMENT AND CONTRIBUTIONS

1.1 Statement of the Problem

1.1.1 Statement of the Biotechnological Problem

Mass spectrometry (MS) imaging has emerged as a powerful tool for revealing the spatial distribution of molecules in a sample. Using a mass spectrometer with a computer-controlled stage, mass spectra are collected across the surface of the sample, creating a hyperspectral image where each pixel is associated with a corresponding mass spectrum. The mass spectral intensities at a particular m/z-value can then be plotted to create a false-color image showing the relative spatial abundance of that molecule. Using leading technologies such as *matrix-assisted laser desorption/ionization* (MALDI) and *desorption electrospray ionization* (DESI), MS imaging is being applied to a wide range of biological applications including molecular histology of tissue, whole body sections, and bacterial films [1], and biomedical applications such as cancer diagnosis [2, 3]. There are even emerging non-biological applications such as minerals, circuit boards, and fingerprints [1]. However, even as the technology evolves toward ever-increasing mass and spatial resolutions, including 3D applications, the tools available for the downstream workflow and analysis have not advanced at nearly the same pace as the rapid instrumentrational improvements. There is a dearth of freely-available, open-source tools that can handle the cutting-edge MS imaging datasets being collected today, and those that exist often lack statistical analysis methods appropriate to the experimental design and data structure. Therefore, there is a pressing need for statistically-minded software that solves the problems of user-accessible visualization, processing, and analysis of MS imaging experiments.

1.1.2 Statement of the Statistical and Computational Problem

MS imaging experiments can be thought of as a "data cube" of spectral intensities, with x and y spatial dimensions, and an additional dimension representing the m/z-values. For experiments with multiple samples or 3D imaging experiments, this becomes a data hypercube, with additional dimensions for a spatial z-axis, the sample IDs, or even a time axis for time-course experiments. Because technological improvements are being made in both mass and spatial resolutions, this means the size of datasets is growing exponentially. Statistical methods designed for analysis of MS imaging experiments must scale to both a large number of features P (i.e., the m/z-values) and a large number of observations N (i.e., the number of pixels across all samples, subjects, time points, etc.), while simultaneously respecting the spatial (and possible temporal) structure of the data. While smaller datasets may be loaded into computer memory, high-resolution datasets quickly grow to sizes that exceed the memory limits of a single machine. The complex structure of the data creates additional challenges both statistically, in modeling the spatial correlation appropriately, and computationally, in that the most appropriate algorithms may not be easily parallellized, and the data cannot be partitioned in a straightforward manner. Visualization and exploratory data analysis are already difficult considering the high-dimensionality and complexity of the data, and become even more difficult when considering MS imaging experiments with multiple samples. Interactive graphics are a practical necessity for visualizing such datasets. The goal of this work is to solve a subset of these problems which are currently feasible – such as proposing spatially-aware statistical methods and providing software for statistical analysis – and elsewhere provide a framework to greatly lower the barrier to entry for working with MS imaging datasets so that other statisticians and computational scientists may more easily make contributions to this exciting field – such as by providing software that simplifies development of statistical methods for larger-than-memory datasets.

1.2 Statement of the Contributions

1.2.1 Statistical Methods

Two of the primary goals of current MS imaging experiments are classification (for supervised experiments) and segmentation (for unsupervised experiments). For MS imaging experiments, classification involves the assignment of pixels to known classes, and segmentation involves the assignment of pixels to discovered segments. A common secondary goal is the selection of a subset of important features that define the classes or segments. I proposed the *spatial shrunken centroids* framework for performing classification and segmentation of MS imaging experiments with feature selection [4]. The proposed spatial shrunken centroids framework:

- combines the desirable properties of *nearest shrunken centroids* [5,6] and *spatially-aware clustering* [7] by using statistical regularization to select important features, while also using locally-adaptive spatial smoothing to improve the resulting classified or segmented image.

- allows data-driven selection of an appropriate number of segments in unsupervised experiments, by taking advantage of the empirical relationship between sparsity in the number of selected features and the predicted number of segments in the resulting segmentations.

- facilitates interpretation of the segmented or classified images by automatically selecting features that best differentiate each class or segment from the others.

- enables the characterization and visual inspection of the uncertainty in class or segment membership by calculating their probabilities through analogy to Gaussian mixture models.

The proposed framework is applicable to and has been tested on both DESI- and MALDI-imaging datasets, including both supervised and unsupervised experiments.

1.2.2 Open-source Software and Implementation

To solve the problem of the lack of freely available software tools offering scalable statistical analysis of MS imaging experiments, I contributed two major software packages for statistical computing with MS imaging experiments.

Cardinal

I designed and implemented the R software package *Cardinal* (`www.cardinalmsi.org`) [8], which provides a full pipeline for the import, pre-processing, visualization, and statistical analysis of MS imaging experiments. As the first R package designed specifically for the analysis of MS imaging experiments, *Cardinal* implements both statistical methods and computational infrastructure, including:

- native support for importing two most common MSI data formats (Analyze 7.5 and imzML).
- efficient, modular data structures for working with biological imaging data such as
  - `iSet` an extensible virtual class for imaging experiments
  - `MSImageSet` a class for MSI experimental data and metadata
  - `MSImageData` a class for efficient storage of MSI data
  - `MSImageProcess` a class for tracking pre-processing applied to MSI data
  - `IAnnotatedDataFrame` a class for pixel-level metadata
  - `MIAPE-Imaging` a class for the Minimum Information about a Proteomics [Imaging] Experiment, based on the imzML specification [9]
  - `ResultSet` a class for the results of analysis of imaging experiments
- `generateSpectrum` and `generateImage` functions allowing highly customizable simulation of mass spectra and images for testing new analytic methods.

- `pixelApply` and `featureApply` methods allowing for the easy application of arbitrary functions over the data of imaging experiments (similar to `lapply`), including optionally applying functions over specific conditions (similar to `tapply`) such as the sample ID.
- powerful and flexible visualization tools for plotting mass spectra, molecular ion images, and analysis results, including a formula interface based on **lattice** graphics allowing conditional plots arranged in a trellis display or overlaid using transparency.
- image processing tools including contrast enhancement and spatial smoothing.
- spectral processing tools including normalization, smoothing, baseline reduction, binning and resampling, and peak picking and peak alignment (all implemented using `pixelApply`).
- statistical methods including *spatial shrunken centroids*, *spatially-aware clustering*, *principal components analysis* (PCA), *projection to latent structures* (PLS), and *orthogonal projection to latent structures* (O-PLS) for the analysis of imaging experiments.

Cardinal (v1.5.0) consists of 7,870 lines of R code and 1,757 lines of C and C++ code. It has been downloaded more than 1,800 times from distinct IP addresses in the four months since its public release with Bioconductor 3.1 on April 17, 2015, and has won the John M. Chambers Statistical Software Award for 2015. *Cardinal* is open source and freely available from Bioconductor at `www.bioconductor.org/packages/Cardinal`. Its source code is hosted on Github at `www.github.com/kuwisdelu/Cardinal`.

matter

To enable *Cardinal* to scale to high-resolution, high-throughput MS imaging experiments, as well as facilitate development of new statistical methods for MS imaging experiments and other scientific domains with larger-than-memory datasets, I designed and implemented the *matter* R package. *matter* provides computational infrastructure for statistical computing with data on disk. Its functionalities include:

- flexible definitions of the structure of on-disk binary data, customizable to different domain-specific file formats, and allowing for a single experimental dataset to span multiple files
- efficient data structures in R and C++ for accessing on-disk data, including:
  - `atoms` an S4 class, defined in R, to refer to a set of contiguous ('atomic') sectors of disk belonging to the dataset, with known byte offsets and extents
  - `matter` an S4 class, defined in R, which translates a combination of `atoms` objects into a vector or matrix
  - `Atoms` a C++ class for on-demand reading of data on disk from the sectors described by an `atoms` object, possibly from non-contiguous disk locations, batched into sequential reads for efficiency whenever possible
  - `Matter` a C++ class for translating the data read from disk (by an `Atoms` object) into an R-friendly data format
  - `MatterAccessor` a C++ class for accessing and iterating over a buffered version of a `Matter` vector or over a row or column of a `Matter` matrix.
- efficient indexing into `matter` on-disk vectors and matrices without loading the full object into memory
- an `apply` method (similar to R's built in `apply` function) for iterating over rows and columns of `matter` matrices without loading more than a single row or column into memory at a time
- transposition of `matter` matrices without loading any data into memory or touching a single byte of data on disk

- basic linear algebra for `matter` on-disk matrices with R in-memory matrices
- calculation of statistical summaries such as mean and variance, for vectors, and for rows and columns of matrices, without fully loading the object into memory
- linear regression and fitting of generalized linear models without loading the full dataset into memory using the *biglm* package
- principal components analysis without loading the full dataset into memory using the *irlba* package

matter (v.0.4.0) consists of 1,345 lines of R code and 1,664 lines of C and C++ code. Portions of code using the same ideas as *matter* were incorporated into *Cardinal* v1.4, which was released as part of Bioconductor 3.3 on May 4, 2016. *matter* will be submitted to Bioconductor for the October 2016 release of Bioconductor 3.4, and *Cardinal* will transition to using *matter* as its primary backend for larger-than-memory datasets. The source code of *matter* is hosted on Github at `www.github.com/kuwisdelu/matter`.

1.3 Outline

This dissertation is organized as follows. Chapter 1 introduces the biotechnological and statistical problem and summarizes the contributions. Chapter 2 describes the experimental procedures of MS imaging, the spectral and image processing steps leading up to analysis, and the existing statistical methods, computational algorithms, and software for MS imaging. Chapter 3 describes the spatial shrunken centroids statistical method for supervised classification and unsupervised segmentation of MS imaging experiments. Chapter 4 evaluates the proposed spatial shrunken centroids method by comparing it to existing methods based on results from experimental datasets. Chapter 5 describes the R package *Cardinal* for the processing, visualization, and analysis of MS imaging experiments. Chapter 6 describes the R package *matter* for statistical analysis of larger-than-memory datasets (including MS imaging datasets) on disk. Chapter 7 summarizes the proposed statistical methods and software and discusses the directions for future work.

2. BACKGROUND

2.1 MS-based Imaging Experiments

2.1.1 Introduction

Recently, MS imaging has been shown to be promising in a wide range of biological applications such as molecular histology of tissue, whole body sections, and bacterial films [1], and biomedical applications such as cancer diagnosis [2, 3]. MS imaging visualizes the spatial distribution of molecular ions in a sample by repeatedly collecting mass spectra across its surface. This dissertation will focus on statistical methods and statistical computing environments for the analysis of MS imaging experiments.

This section discusses the structure and steps of MS imaging experiments up to the point of statistical analysis. MALDI and DESI are the leading technologies for performing MS imaging. MALDI-imaging requires the application of a matrix solution, and is typically used to detect large molecules such as peptides and proteins. DESI-imaging does not require a matrix, and is typically used to detect small molecules such as lipids, metabolites, and drug molecules [10]. Equipped with a computer-controlled sample stage, a mass spectrometer rasters across the sample, and collects individual mass spectra from discrete or continuous locations. The intensities at a particular m/z value and spatial location can then be plotted as pixels in a false-color image, called an *ion image*, that displays the spatial distribution of the analyte associated with that m/z value.

Computational analysis of MS imaging experiments typically consists of processing, followed by a statistical analysis [11]. The processing ensures that mass spectral intensities are comparable across all mass spectra in the experiment. This is typically done via normalization and (if necessary) baseline reduction. Furthermore, the processing extracts spectral features that correspond to the underlying analytes. This is typically done via peak picking, or m/z binning or resampling. Much progress has already been made in the processing of MS imaging data. Many mature tools for processing mass spectra already exist [12]. However, a major bottleneck is the downstream statistical analysis of the processed data.

Two common primary goals of statistical analysis of MS imaging experiments post signal processing are classification (for supervised experiments), i.e. assigning pixels to pre-defined classes based on their spectral profiles, and segmentation (for unsupervised experiments), i.e. assigning pixels to newly discovered segments with relatively homogenous and distinct spectral profiles.

However, achieving these goals is often quite difficult due to the large and complex nature of the datasets, and due to the biological and technical variation in intensities of spectral features. While a number of machine learning algorithms for analysis of MS imaging experiments exist, methods for statistical inference are key for distinguishing the systematic signals in the spectra from noise. This dissertation focuses on the downstream statistical analysis steps, which take place after the detection, quantification, alignment, normalization, and (optionally) identification of the initial set of high quality spectral features.

2.1.2 Overview of Spectral Processing Steps

Processing of mass spectra ensures that mass spectral intensities are comparable across all spectra in the experiment. The processing steps described below are all implemented in *Cardinal* as discussed in Chapter 5.

- *Normalization* ensures that mass spectra from different pixels are comparable to each other. This is typically done by equalizing the sum of all intensities to a common total ion current (TIC), but normalization to a known reference is also possible. More experimental work comparing different methods of normalization for MS imaging is still required [13].

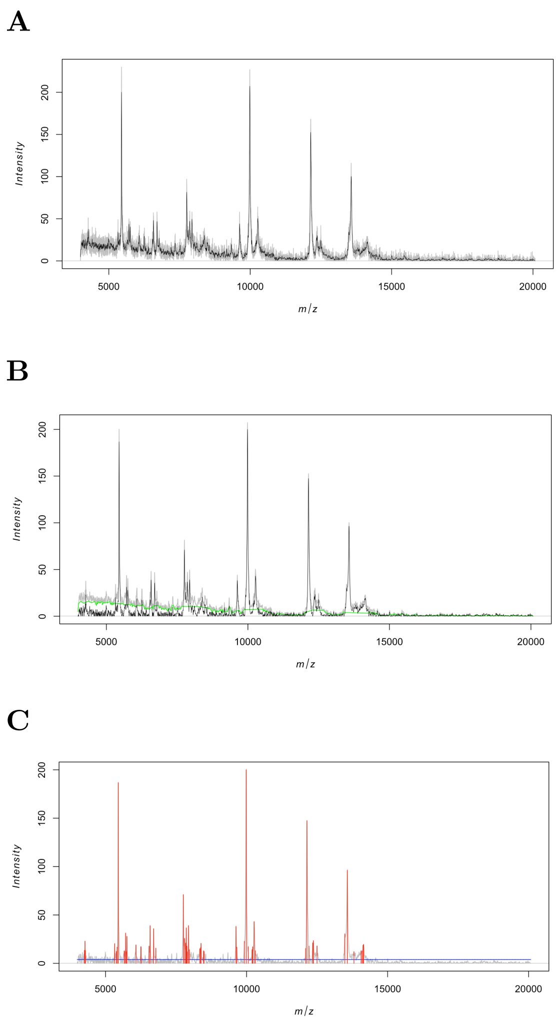


Fig. 2.1. **Spectral processing.** *A*, signal smoothing, *B*, baseline reduction, and *C*, peak picking, as implemented in *Cardinal* .

- *Signal smoothing* (shown in Figure 2.1A) reduces noise in the mass spectral signal for downstream baseline reduction and peak picking. This can be done using techniques such as moving average filter, Savitsky-Golay filter, Gaussian filter, etc. [12]
- *Baseline reduction* (shown in Figure 2.1B) removes unwanted background from the mass spectra (usually due to the matrix in MALDI). The baseline is typically

estimated using linear interpolation of minima or medians, or LOESS regression, and then subtracting off the estimated baseline [12].

- *Peak picking* (shown in Figure 2.1C) extracts spectral features corresponding to biological signal (i.e., analytes). There are many mature algorithms for detection of peaks in mass spectra based on various criteria, including signal-to-noise ratio (SNR), slopes of peaks, local maxima, peak shape, peak width, etc. [12, 14]

- *Peak alignment* corrects for unwanted shifts in m/z values to ensure peaks are comparable across all spectra, for example, by aligning the detected peaks in each spectrum to peaks in the mean spectrum or other reference [13].

- *Binning* and *resampling* are options to reduce the size of the dataset prior to analysis or other spectral processing steps.

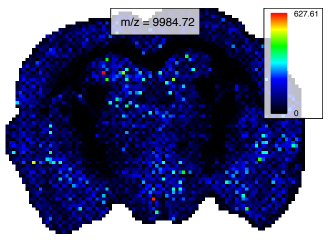
2.1.3 Overview of Image Processing Steps

Processing of molecular ion images (shown in Figure 2.2A) is often necessary due to the multiplicative noise prevalent in MS imaging experiments. The steps typically consist of:

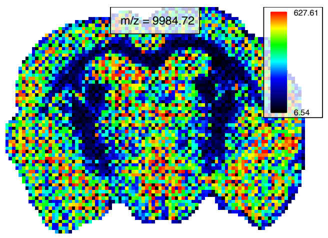
- *Spatial smoothing* (shown in Figure 2.2B) reduces noise while retaining and highlighting the important spatial patterns in the image, using methods such Gaussian filter or adaptive bilateral filter [15].

- *Contrast enhancement* (shown in Figure 2.2C) corrects unbalanced contrast in an image (usually from multiplicative noise) that may hide spatial patterns, using methods such as histogram equalization or suppression of the brightest pixels [11].

A



B



C

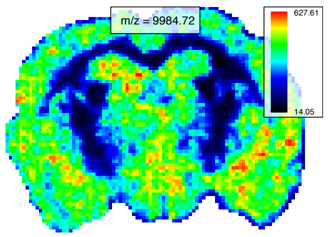


Fig. 2.2. **Image processing.** *A*, a raw molecular ion image. *B*, an ion image with contrast enhancement. *C*, an ion image with contrast enhancement and spatial smoothing.

2.2 Review of Related Work

2.2.1 Review of Existing Statistical Methods for Analysis of MS Imaging Experiments

Traditional multivariate statistical methods are frequently used for both classification and segmentation. For classification, methods including *linear discriminant* *analysis* (LDA), *projection to latent structures discriminant analysis* (PLS-DA) and *orthogonal projection to latent structures discriminant analysis* (O-PLS-DA) have proven effective [2, 3, 16–19]. For segmentation, clustering methods such as hierarchical clustering or k-means (sometimes preceded by *principal components analysis* (PCA) to reduce the dimensionality of the spectra) are frequently used [20–22]. The traditional multivariate methods have two drawbacks. First, they are agnostic of the spatial structure of the data. They treat each pixel independently, and ignore similarities of spectra acquired from spatially proximate locations, thereby compromising the accuracy of the results. Second, they do not reduce the input features to more informative subsets, thereby compromising the interpretation.

Although statistical regularization has become a method of choice for extracting subsets of informative features from highly multivariate data, most such methods have not yet been applied to MS imaging experiments. One such method is *nearest shrunken centroids* introduced by Tibshirani *et al.* [5, 6], which was originally developed for classification of gene expression microarrays. A related method has been applied to classify tissues in MS imaging experiments using regularized logistic regression [23]. However, similarly to the multivariate analysis methods, they do not account for the spatial structure of the data.

2.2.2 Review of Existing Computational Algorithms for Analysis of MS Imaging Experiments

Several recent computational algorithms were specifically designed to account for the spatial structure of MS images. One family of methods relies on the spatial structure to detect quality peaks from raw spectra [24, 25]. Although highly valuable, these methods stop at processing the signals, and do not address the goals of image segmentation or image classification. Another family of methods, including *spatially-aware clustering* and *spatially-aware structurally-adaptive clustering* by Alexandrov and Kobarg [7], account for the spatial structure of the data, and demonstratively improve the quality of image segmentation [11]. However, similarly to the multivariate analysis methods, they do not select subsets of spectral features that define the segments, and rely on *post hoc* techniques to interpret the features associated with the segments, e.g. using the Pearson correlation between a segment and the single ion images [11, 20].

Most existing computational algorithms designed for analysis of MS imaging experiments are not statistical in nature, and do not allow for statistical inference.

2.2.3 Review of Existing Software for MS Imaging

Existing freely-available software for MS imaging include BioMap, DataCube Explorer, and MSiReader, and typically focus on data exploration and visualization, without emphasis on statistical modeling and inference. They generally require that the full dataset must fit into computer memory. Commercially-available software such as SCiLS Lab (SCiLS), flexImaging (Bruker), HDI (Waters), and TissueView (AB Sciex) sometimes include more advanced analytic capabilities (particularly SCiLS), sometimes including support for larger-than-memory datasets (again SCiLS), but are often expensive and are not open-source. In addition, most existing software for MS imaging are designed around exploration or analysis of a single sample, rather than experiments involving multiple samples.

3. SPATIAL SHRUNKEN CENTROIDS METHOD FOR MODEL-BASED CLASSIFICATION AND SEGMENTATION OF MS IMAGES

3.1 Overview of Spatial Shrunken Centroids

Spatial shrunken centroids is a statistical model-based framework for both supervised classification and unsupervised segmentation. It takes as input a set of previously detected, quantified, aligned and normalized features, produced by any signal processing method(s) of choice. It combines the advantages of both spatially-aware clustering by Alexandrov and Kobarg [7] and statistical regularization by Tibshirani *et al.* [5,6].

In Chapter 4, we will show that for unsupervised segmentation, the spatial probabilistic modeling provides better quality segmentation. It characterizes the probability of segment membership for each pixel, and allows us to quantify and visualize the uncertainty of segmentation for each pixel. Moreover, statistical regularization aids interpretation by automatically selecting subsets of the spectral features, such that each subset defines each segment. Statistical regularization also enables data-driven selection of the number of segments. Similarly, for supervised image classification probabilistic modeling characterizes the probability of pre-defined tissue class membership for each pixel, and aids interpretation by automatically selecting subsets of spectral features that define each class.

3.2 Proposed Statistical Framework for Supervised Classification and Unsupervised Segmentation of Mass Spectrometry Images

Let $m = 1, \dots, M$ denote the index of the biological sample, i.e. a slide with one (or several) tissues. On slide m, the experiment collects N_m spectra at N_m total pixel locations. Therefore, over all the samples, the experiment contains $N = \sum_{m=1}^{M} N_m$ spectra and pixels.

Let (i, j) denote the location of a pixel on a sample m. We do not assume that the samples are rectangular in shape, so the indices (i, j) are arbitrary. However, we do assume $(i+\delta_i, j+\delta_j)$ describes the location of a pixel (δ_i, δ_j) away on the same sample. We assume that the spectra acquired at these locations have been processed, so that every spectrum has the same P features, defined as a picked peak or a binned m/z range. We also assume that the pixel intensities are normalized, so that spectra are comparable across pixels and across samples. Then, denote the spectrum acquired at a pixel location (i, j) on sample m as $\mathbf{x}_{ijm} = \{x_{ijmp},\ p = 1, \dots, P\}$. In other words, spectrum $\mathbf{x}_{ijm}$ is a vector of scalar intensities x_{ijmp} for P spectral features.

Suppose also that the spectra and the pixels belong to one of K classes (for supervised classification), or segments (for unsupervised segmentation). For supervised classification, the class membership is known, for example, from annotation by a pathologist, and the statistical goal of the experiment is to classify each pixel to one of these classes in a supervised manner based on its spectrum. Alternatively, for unsupervised segmentation, the class membership is unknown, and the statistical goal of the experiment is to discover these classes from the spectra in an unsupervised manner. Let N_k denote the number of spectra, and the number of pixel locations, assigned to class $k = 1, \dots, K$ by an unsupervised or a supervised procedure.

Additionally, we denote the mean spectrum for a known class or discovered segment k as $\bar{\mathbf{x}}_k$, and the overall mean spectrum as $\bar{\mathbf{x}}$. That is, $\bar{\mathbf{x}}_k$ is a vector of P scalar intensities $\bar{x}_{kp}$, which are the mean intensities for spectral feature p, over spectra from all pixel locations assigned to class k, and $\bar{\mathbf{x}}$ is a vector of P scalar intensities $\bar{x}_p$, which are the overall mean intensities for spectral feature p, over all spectra.

In the following section we discuss the proposed spatial shrunken centroids framework for both supervised classification and unsupervised segmentation. For supervised classification, the method relies on the known classes. For unsupervised segmentation, where the segments are unknown, the segments are initialized randomly or by another segmentation method such as spatially-aware clustering [7], and are updated over multiple iterations until one of several convergence criteria is met. We detail the important steps below. The full algorithms are available in Section 3.3.1 and Section 3.3.2.

3.2.1 Characterization of Classes and Segments by their Shrunken Centroids

In mass spectrometry imaging, a tissue region, condition, or class is typically summarized by a mean spectrum, also called its centroid, $\bar{\mathbf{x}}_k$. Here we propose that each class (or segment) is better represented using shrunken centroids, from the method of nearest shrunken centroids by Tibshirani et al. [5, 6]. This will allow us to compare the class (or segment) centroids to the overall centroid, and to select the informative spectral features (defined as being very dissimilar to the overall centroid). We detail this below.

We follow Tibshirani et al. and calculate the class (or segment) centroids $\bar{\mathbf{x}}_k$, and use statistical regularization to shrink the centroids toward the overall centroid $\bar{\mathbf{x}}$. We then calculate the t-statistic for spectral feature p for class (or segment) k as

$$t_{kp} = \frac{\bar{x}_{kp} - \bar{x}_p}{\hat{\tau}_p \cdot \sqrt{\frac{1}{N_k} - \frac{1}{\sum_{k=1}^{K} N_k}}} \tag{3.1}$$

Here, $\hat{\tau}_p$ is the pooled estimate of the within-class standard deviation for feature p. The number $\sqrt{\frac{1}{N_k} - \frac{1}{\sum_{k=1}^{K} N_k}}$ makes the denominator equal to the estimated standard error of the numerator. Second, we apply the soft thresholding operator $()_+$ to shrink the t-statistics toward 0

$$t'_{kp} = \text{sign}(t_{kp})(|t_{kp}| - s)_+, \quad \text{where } t_+ = t \text{ if } t > 0, \text{ and } t_+ = 0 \text{ if } t \leq 0 \tag{3.2}$$

and s is the shrinkage parameter. Larger values of s lead to a larger number of t-statistics t'_{kp} to be set to 0. Finally, we define the intensities of the shrunken centroids for each feature p for each class (or segment) k as

$$\bar{x}'_{kp} = \bar{x}_p + t'_{kp}\hat{\tau}_p \cdot \sqrt{\frac{1}{N_k} - \frac{1}{\sum_{k=1}^{K} N_k}} \tag{3.3}$$

so that $\bar{\mathbf{x}}'_k = \{x'_{kp},\ p = 1, \ldots, P\}$ is the shrunken centroid for class (or segment) k.

The shrunken centroids $\bar{\mathbf{x}}'_k$ here can be viewed as adjusted mean spectra of the K classes (or segments), where the intensities have been adjusted toward the overall mean spectrum. Therefore, the characteristic mean spectrum for a class (or segment) should differ from the overall mean spectrum only for those spectral features that are truly characteristic of the class (or segment). Spectral features which are not meaningfully different from the overall mean spectrum will have intensities set to the overall mean intensity for that feature.

3.2.2 Selection of Informative Features

The shrunken t-statistics t'_{kp} calculated in Equation 3.2 are well suited for selecting informative features. The spectral features with $t'_{kp} > 0$ are systematically enriched for class (or segment) k. Likewise, spectral features with $t'_{kp} < 0$ are systematically absent from class (or segment) k, as compared to the overall mean spectrum. Spectral features with $t'_{kp} = 0$ are non-informative, as only the features with $t'_{kp} \neq 0$ matter when assigning a pixel's spectrum to class (or segment) k.

3.2.3 Spatially-aware (SA) and Spatially-aware Structurally-adaptive (SASA) Distances

To classify the individual pixel, or to assign a pixel to a segment, we need to define a distance between the spectra from individual pixels and the shrunken centroids. We propose to use the spatially-aware distance defined by Alexandrov and Kobarg [7]. We detail this method below, and show how we adapt it in Section 3.2.4.

Alexandrov and Kobarg proposed a spatially-aware distance between two spectra $\mathbf{x}_{ijm}$ and $\mathbf{x}_{i'j'm}$, which depends on the spectra from pixels within a neighborhood of (i, j) and (i', j'). The authors showed that this approach is beneficial, as it produces better quality segmentations, as compared to naïve methods that do not account for the spatial relationships between pixels [26]. Therefore, for a neighborhood radius of r, the distance between two spectra is defined as

$$d(\mathbf{x}_{ijm},\ \mathbf{x}_{i'j'm}) = \sum_{-r\leq\delta_i,\delta_j,\leq r} \alpha_{\delta_i\delta_j}(\mathbf{x}_{ijm},\ \mathbf{x}_{i'j'm'}) \cdot \|\mathbf{x}_{(i+\delta_i)(j+\delta_j)m} - \mathbf{x}_{(i'+\delta_i)(j'+\delta_j)m'}\|^2 \quad (3.4)$$

Here the $\alpha_{\delta_i\delta_j}(\mathbf{x}_{ijm},\ \mathbf{x}_{i'j'm'})$ are spatial weights of the neighbors. The exact definition of these weights results in either a spatially-aware (SA) distance or a spatially-aware structurally adaptive (SASA) distance. In the SA distance, the weights are defined as

$$\alpha_{\delta_i\delta_j} = \exp\left\{ -\frac{\delta_i^2 + \delta_j^2}{2\sigma^2} \right\}, \quad \text{where } \sigma = (2r+1)/4 \quad (3.5)$$

which are Gaussian weights independent of the spectra and only depend on the neighborhood. Using Gaussian weights, which decrease with the distance $\delta_i^2 + \delta_j^2$ from the neighborhood center, is a natural choice, because it assumes that pixels further away from each other are less related than pixels that are closer together. In the SASA distance, the weights are defined as

$$\alpha_{\delta_i\delta_j}(\mathbf{x}_{ijm},\ \mathbf{x}_{i'j'm'}) = \exp\left\{ -\frac{\delta_i^2 + \delta_j^2}{2\sigma^2} \right\} \cdot \sqrt{\beta_{\delta_i\delta_j}(\mathbf{x}_{ijm})\beta_{\delta_i\delta_j}(\mathbf{x}_{i'j'm'})} \quad (3.6)$$

where

$$\beta_{\delta_i\delta_j}(\mathbf{x}_{ijm}) = \exp\left\{ -\frac{1}{2\lambda^2} \|\mathbf{x}_{(i+\delta_i)(j+\delta_j)m} - \mathbf{x}_{ijm}\|^2 \right\} \quad (3.7)$$

which are adaptive weights that downweight neighborhood locations where the spectra are very different from the neighborhood center. This is designed to preserve edges between morphological regions and small details in local structure, which could otherwise be lost due to oversmoothing by the ordinary Gaussian weights. The term λ is set empirically to the half of the norm of the difference between the two most differing spectra in the neighborhood.

3.2.4 Defining the SA and SASA Distances to the Shrunken Centroid of a Class or Segment

The distance above can be adapted to express the distance between the individual pixels and the shrunken centroids as follows:

$$d(\mathbf{x}_{ijm},\ \bar{\mathbf{x}}'_k) = \sum_{-r\leq\delta_i,\delta_j,\leq r} \alpha_{\delta_i\delta_j}(\mathbf{x}_{ijm}) \cdot \|\mathbf{x}_{(i+\delta_i)(j+\delta_j)m} - \bar{\mathbf{x}}'_k\|^2 \tag{3.8}$$

where defining the $\alpha_{\delta_i\delta_j}$ using the Gaussian weights as in Equation 3.5 results in our version of the SA distance, and using adaptive weights defined as

$$\alpha_{\delta_i\delta_j}(\mathbf{x}_{ijm}) = \exp\left\{ -\frac{\delta_i^2+\delta_j^2}{2\sigma^2} \right\} \cdot \beta_{\delta_i\delta_j}(\mathbf{x}_{ijm}) \tag{3.9}$$

with $\beta_{\delta_i\delta_j}(\mathbf{x}_{ijm})$ as in Equation 3.7 results in our version of the SASA distance. We normalize the weights in both cases so that they sum to 1.

Unlike in Equation 3.4 above, in Equation 3.8 we consider the dissimilarity between a pixel's spectrum and a class (or segment), rather than the dissimilarity between the spectra at two pixels. Note that our version of the SASA distance has only one $\beta_{\delta_i\delta_j}$ rather than two, reflecting this difference. In the case of supervised classification, we will use this distance to classify pixels according their spectrum's similarity to the shrunken centroids of known classes. In the case of unsupervised image segmentation, we will use this distance to iteratively update the pixels assigned to discovered segments.

Note also that in both supervised and unsupervised situations this requires the empirical selection of the shrinkage parameter s. Moreover, for unsupervised segmentation, the selection of the number of segments K is also required. The procedure for selecting these parameters and their effect and implications will be described further in Section 3.2.6.

3.2.5 Assignment of Segment or Class Probabilities to Pixels

For supervised classification nearest shrunken centroids can be interpreted as a regularized version of *linear discriminant analysis* [5, 6]. In this case each of the K classes has a prior probability π_k, and is modeled as a multivariate Gaussian distribution. All classes are assumed to share a common diagonal within-class covariance matrix. This leads to a straightforward way to calculate probabilities for individual observations belonging to a class using Gaussian likelihoods. By analogy, we calculate a discriminant score based on the SA or SASA distances from each spectrum $\mathbf{x}_{ijm}$ to each of the shrunken centroids $\bar{\mathbf{x}}_{\mathbf{k}}$ as

$$\mathcal{D}(\mathbf{x}_{ijm},\ \bar{\mathbf{x}}_k') = \frac{1}{\hat{\tau}_p^2}\, d(\mathbf{x}_{ijm},\ \bar{\mathbf{x}}_k') - 2\ \log \pi_k \tag{3.10}$$

where, as before, $\hat{\tau}_p$ is the pooled within-class standard deviation for feature p. We typically estimate the prior probabilities empirically as $\hat{\pi}_k = N_k/N$. If the training data is not representative of the population, different priors could be used. Because we are using spatial distances which incorporate spectra from multiple pixels, the discriminant scores cannot be interpreted directly as following Gaussian distributions. However, we empirically demonstrate below that the technique still produces good results in practice. Therefore, we further follow Tibshirani et al. by calculating class probabilities for each spectrum $\mathbf{x}_{ijm}$ for each class (or segment) k as

$$\hat{p}_k(\mathbf{x}_{ijm}) = \frac{e^{-(1/2)\mathcal{D}(\mathbf{x}_{ijm},\, \bar{\mathbf{x}}_k')}}{\sum\limits_{l=1}^{K} e^{-(1/2)\mathcal{D}(\mathbf{x}_{ijm},\, \bar{\mathbf{x}}_l')}} \tag{3.11}$$

A pixel is assigned to the class with the highest $\hat{p}_k(\mathbf{x}_{ijm})$.

Unsupervised segmentation follows the same procedure. K is the maximum number of segments, and the segments are initialized randomly or with another segmentation procedure. We typically use $\pi_k = 1/K$, but a semi-supervised procedure could be developed which uses different priors and a different initialization procedure. During each iteration of the segmentation, a pixel is updated as belonging to the segment with the highest $\hat{p}_k(\mathbf{x}_{ijm})$ using Equation 3.11.

3.2.6 Selection of Parameters

The proposed framework requires the choice of the shrinkage parameter s, and, for unsupervised segmentation, the number of segments K.

In the case of supervised classification, the classes are known, and therefore s can be selected by cross-validation. Specifically, given a set of M biological samples on M slides, each slide is viewed as an experimental unit for cross-validation. For experiments with a small number of biological replicates, M-fold (i.e., leave-one-sample-out) cross-validation can be performed. Within each fold (or sample) of cross-validation, fit spatial shrunken centroids for a range of values of s. The final selected value of s is the one that maximizes the overall classification accuracy on the left-out samples. This is illustrated for the human RCC experiment in Figure 4.15.

In the case of unsupervised segmentation, the individual segments, and also the exact number of segments, are unknown. However, there is a relationship between the number of informative features in the model, expressed by the shrinkage parameter s, and the number of segments K. First, spurious segments tend to be defined by non-informative features. When those are removed through statistical regularization, the spurious segments become empty. They have $N_k = 0$, and are, in fact, removed. Second, excessive regularization can remove some informative features, and this results in the loss of the correct segments. We balance the regularization and the number of segments by creating segmentations for multiple values of s and K, and then plotting the relationship between s and the number of non-empty segments. We illustrate this using experimental data in Section 4.2.2 and in Section 4.2.2.

3.3 Algorithm and Implementation

The full algorithms for spatial shrunken centroids for a single set of parameters for supervised classification and for unsupervised segmentation are described below. They are implemented in the R package *Cardinal* (`cardinalmsi.org`) [8], which is described in Section 5.

The implementation is efficient, utilizing C and C++ for speed. It can efficiently handle large datasets, and is limited only by the requirement that the dataset must be fully loaded into memory. For example, the segmentations for the fetal pig dataset (143 peaks and 4,959 pixels) in Figure 4.7E and Figure 4.7F took 51 and 52 seconds, respectively. The segmentations for the cardinal painting (51 peaks and 12,600 pixels) in Figure 4.8E and Figure 4.8F took 67 and 48 seconds, respectively. The cross-validation for the human RCC dataset (850 features and 6,077 pixels) in Figure 4.15 took 69 seconds.

3.3.1 Procedure for Spatial Shrunken Centroids Classification (Supervised)

The following describes the algorithm for a single set of parameters for a single fold of cross-validation. Parameters should be selected as described in Section 3.2.6.

Input

1. Training set of samples with class labels $k = 1, \dots, K$
2. Testing set of samples $m = 1, \dots, M$
3. Parameters
   (a) Neighborhood radius r
   (b) Shrinkage parameter s

Fitting – performed on samples from the training set

1. Calculate the overall centroid $\bar{\mathbf{x}}$
2. For each class $k = 1, \ldots, K$:
   (a) Calculate the class centroid $\bar{\mathbf{x}}_k$
   (b) For each feature $p = 1, \ldots, P$:
      i. Calculate the t-statistics t_{kp} [Equation 3.1].
      ii. Calculate the shrunken t-statistics t'_{kp} [Equation 3.2].
   (c) Calculate the class shrunken centroid $\bar{\mathbf{x}}'_k$ [Equation 3.3].
3. Output the shrunken t-statistics t'_{kp} and shrunken centroids $\bar{\mathbf{x}}'_k$.

Prediction – performed on samples from the testing set

1. For each pixel at a location (i, j) on sample $m = 1, \ldots, M$:
   (a) For each class $k = 1, \ldots, K$:
      i. Calculate the distance to the class centroid $d(\mathbf{x}_{ijm},\ \mathbf{x}'_k)$ [Equation 3.8].
      ii. Calculate discriminant score $\mathcal{D}(\mathbf{x}_{ijm},\ \bar{\mathbf{x}}'_k)$ [Equation 3.10].
      iii. Calculate the class membership probability $\hat{p}_k(\mathbf{x}_{ijm})$ [Equation 3.11]
   (b) Assign the pixel to the class with the highest probability $\hat{p}_k(\mathbf{x}_{ijm})$.
2. Output the class assignments and class probabilities $\hat{p}_k(\mathbf{x}_{ijm})$.

3.3.2 Procedure for Spatial Shrunken Centroids Segmentaiton (Unsupervised)

The following describes the algorithm for a single set of parameters. Parameters should be selected as described in Section 3.2.6.

Input

1. Unlabeled samples $m = 1, \ldots, M$

2. Maximum number of iterations *iter.max*
3. Parameters
   (a) Neighborhood radius r
   (b) Maximum number of segments K
   (c) Shrinkage parameter s

Fitting

1. Use SA or SASA clustering by Alexandrov and Kobarg [7] to initialize the K segments.
2. Calculate the overall centroid $\bar{\mathbf{x}}$
3. For each segment $k = 1, \ldots, K$ with $N_k \neq 0$:
   (a) Calculate the segment centroid $\bar{\mathbf{x}}_k$
   (b) For each feature $p = 1, \ldots, P$:
      i. Calculate the t-statistics t_{kp} [Equation 3.1].
      ii. Calculate the shrunken t-statistics t'_{kp} [Equation 3.2].
   (c) Calculate the segment shrunken centroid $\bar{\mathbf{x}}'_k$ [Equation 3.3].
4. For each pixel at a location (i, j) on sample $m = 1, \ldots, M$:
   (a) For each segment $k = 1, \ldots, K$:
      i. Calculate the distance to the segment shrunken centroid $d(\mathbf{x}_{ijm},\ \mathbf{x}'_k)$ [Equation 3.8].
      ii. If a segment has $N_k = 0$, define the distance to it as $d(\mathbf{x}_{ijm},\ \mathbf{x}'_k) = \infty$.
      iii. Calculate discriminant score $\mathcal{D}(\mathbf{x}_{ijm},\ \bar{\mathbf{x}}'_k)$ [Equation 3.10].
      iv. Calculate the segment membership probability $\hat{p}_k(\mathbf{x}_{ijm})$ [Equation 3.11]
   (b) Assign the pixel to the segment with the highest probability $\hat{p}_k(\mathbf{x}_{ijm})$.

5. Update the segments with the pixel assignments from step 4b.

6. Repeat steps 3–5 until no segments change, or at most *iter.max* times.

7. Output the shrunken t-statistics t'_{kp}, shrunken centroids $\bar{\mathbf{x}}'_k$, and probabilities $\hat{p}_k(\mathbf{x}_{ijm})$.

4. EVALUATION AND DISCUSSION FOR SPATIAL SHRUNKEN CENTROIDS

4.1 Datasets for Evaluating Spatial Shrunken Centroids

4.1.1 Unsupervised Segmentation: Pig Fetus Cross-section

The primary goal of this experiment was to discover morphological features of the pig fetus, such as major organs, through unsupervised analysis of the mass spectra. A secondary goal was to find spectral features associated with the morphological features. Figure 4.1A is an optical image of the H&E stained tissue section showing the general morphology of the pig fetus, including major organs such as the brain, heart, and liver.

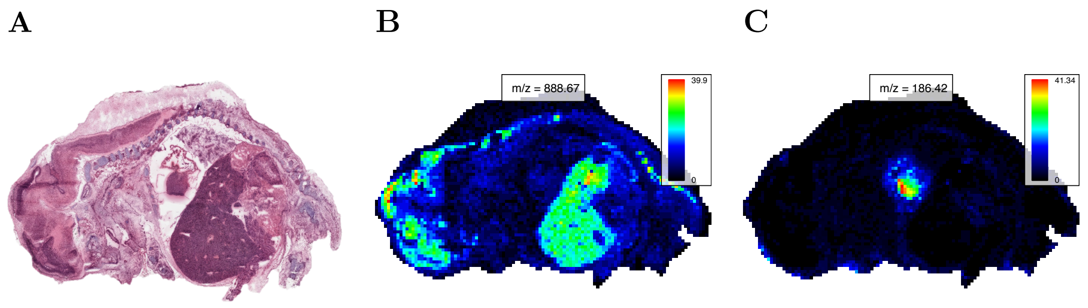


Fig. 4.1. **Pig fetus cross-section: morphology and single ion images.** *A*, Optical image of H&E stained pig fetus cross-section showing its morphology, including the brain (left), heart (center), and liver (dark region below heart). *B–C*, Characteristic ion images for the pig fetus dataset at *B*, 888.67 m/z, showing the brain and liver, and *C*, 186.42 m/z, showing the heart.

Mass spectra were collected using a Thermo Finnigan LTQ linear ion trap mass spectrometer with a DESI ion source over the 150–1,000 m/z range. The images were cropped to remove non-informative spectra originating from the glass slide. The cropped dataset consisted of 4,959 mass spectra with 10,200 spectral features. The mass spectra were normalized to a common total ion current, and peak picking was performed to reduce the dataset to 143 peaks. All data processing and analysis was performed using *Cardinal* [8].

Figure 4.1B shows a single ion image featuring the brain and liver, and Figure 4.1C shows a single ion image featuring the heart. Below, we will use this dataset to demonstrate unsupervised statistical analysis using all the mass spectral peaks to recover the major morphological features.

4.1.2 Unsupervised Segmentation: Cardinal Painting with Known Segmentation

The goal of this experiment was to use a controlled sample to evaluate the quality of data acquisition and statistical analysis. A painting of a cardinal on paper was affixed to a glass slide and MS imaging was applied. An optical image of the cardinal painting during data acquisition is shown in Figure 4.2A.

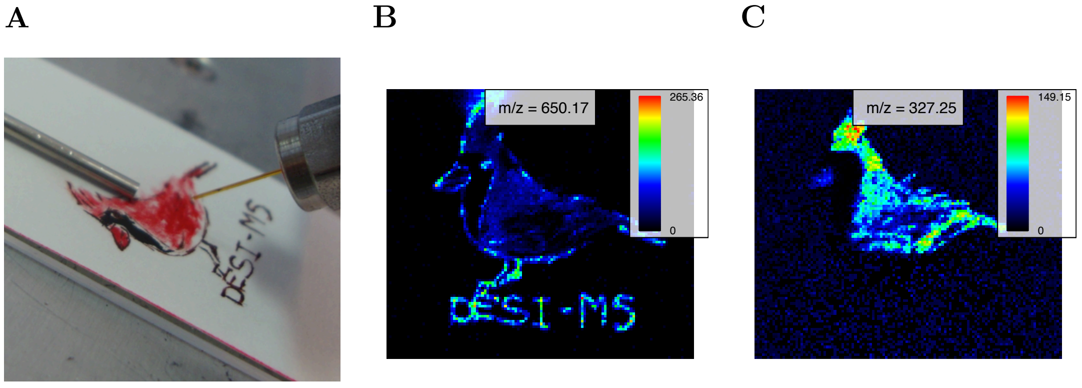


Fig. 4.2. **Cardinal painting: optical image and single ion images.** *A*, Optical image of the cardinal painting during collection of mass spectra. *B–C*, characteristic single ion images for the cardinal painting dataset at *B*, 650.17 m/z, showing the "DESI-MS" text, and *C*, 327.25 m/z, showing the body (red pigment).

The mass spectra were acquired using a Thermo Finnigan LTQ linear ion trap mass spectrometer with a DESI ion source over the 100–1,000 m/z range. The dataset consisted of 12,600 mass spectra with 10,800 spectral features. Mass spectra were normalized to a common total ion current, and peak picking was performed to reduce the dataset to 51 peaks. All data processing and analysis was performed using *Cardinal* .

Figure 4.2B shows a single ion image featuring the "DESI-MS" text part of the painting and Figure 4.2C shows a single ion image featuring the red pigment used in the cardinal body. The painting itself shown in Figure 4.2A can be considered the ground truth image. We use this dataset to evaluate the ability of the unsupervised statistical analysis of the mass spectral peaks to recover the ground truth.

4.1.3 Unsupervised Segmentation: Rodent Brain Images of Varying Quality

The goal for these datasets is to compare the results of several similar experiments of varying data quality. All three experiments involved a rodent brain.

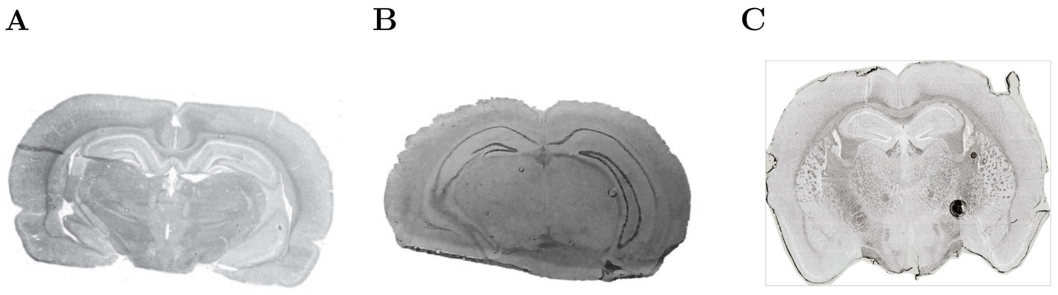


Fig. 4.3. **Rodent brain morphologies.** *A*, Optical image of rat brain (R1). *B*, Optical image of mouse brain (R2). *C*, Optical image of mouse brain (R3).

The first dataset (R1) is a high-quality image of a rat brain [7, 26] which is shown in Figure 4.3A. Mass spectra were acquired on a Bruker Autoflex III MALDI-TOF mass spectrometer over the 2,500 to 25,000 m/z range. The images were cropped to remove non-informative spectra, and only the 2,500 to 10,000 m/z range was used. The reduced dataset consisted of 20,185 mass spectra with 3,045 spectral features. The mass spectra were normalized to a common total ion current, and baseline correction was performed using ClinProTools. *Cardinal* was thereafter used to perform peak picking to reduce the dataset to 80 peaks. Except for baseline correction and normalization, all data processing and analysis was done in *Cardinal* .

The second dataset (R2) is a mouse brain shown in Figure 4.3B. This experiment produced high quality spectra but with a moderate amount of experimental noise. Mass spectra were acquired on a Thermo Finnigan LTQ linear ion trap mass spectrometer with a DESI ion source over the 200–1,000 m/z range. The images were cropped to remove non-informative spectra. The cropped dataset consisted of 8,950 mass spectra with 9,600 spectral features. The mass spectra were normalized to a common total ion current, and peak picking was performed to further reduce the dataset to 123 peaks. All data processing and analysis was done in *Cardinal* .

The third dataset (R3) is a mouse brain shown in Figure 4.3C. This dataset features a high degree of experimental noise. Mass spectra were acquired using an AB Sciex MALDI TOF/TOF 5800 System over the 4,000 to 20,000 m/z range. The images were cropped to remove non-informative spectra. The cropped dataset consisted of 4,923 mass spectra with 22,667 spectral features. The mass spectra were normalized to a common total ion current, smoothed, and baseline corrected. Peak picking was then performed to reduce the dataset to 57 peaks. All data processing and analysis was done in *Cardinal* .

We will use these datasets to characterize the ability of the results of statistical analysis to reflect differences in data quality.

4.1.4 Supervised Segmentation: Human Renal Cell Carcinoma

The goal of this experiment was to classify renal tissue specimens as cancer or normal. In accordance with approved Institutional Review Board protocols at Indiana

University School of Medicine, matched pairs of tissue were collected from human subjects with renal cell carcinoma (RCC), with each pair consisting of cancerous tissue and adjacent normal tissue [2]. Figure 4.4 shows optical images of the eight tissue pairs we analyzed. Each tissue was manually annotated as normal or cancerous by a pathologist. However, the annotations are based on the dominant tissue type for each whole tissue, so some tissues may contain regions from the non-dominant class.

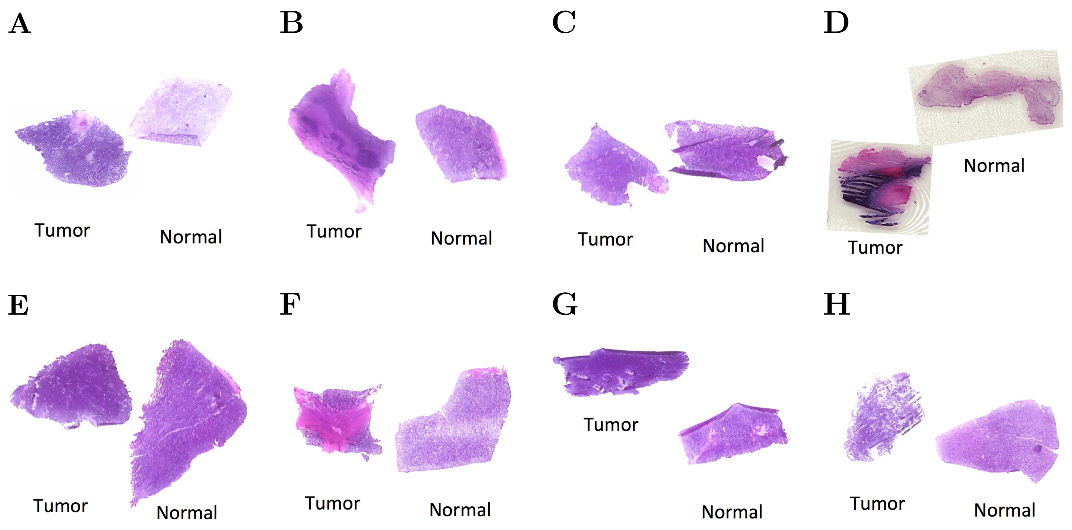


Fig. 4.4. **Human renal cell carcinoma: morphology.** The morphology is characterized by optical images of the H&E stained tissues. For each matched pair, cancerous tissue is on the left, and normal tissue is on the right.

The mass spectra were collected using a Thermo Finnigan LTQ linear ion trap mass spectrometer with a DESI ion source over the 150–1,000 m/z range. The images were cropped to remove non-informative spectra originating from the glass slide. The cropped dataset consisted of 6,077 mass spectra with 10,200 spectral features. Individual tissue samples consisted of between 972 to 3564 mass spectra per matched pair. The mass spectra were normalized to a common total ion current, and resampled to unit resolution resulting in 850 spectral features. All data processing and analysis was performed using *Cardinal* .

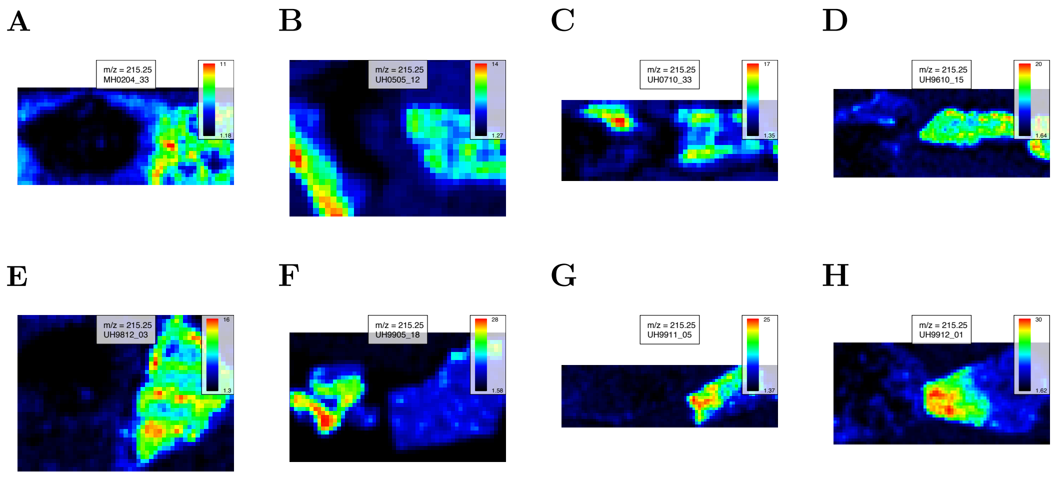


Fig. 4.5. **Human renal cell carcinoma: normal tissue single ion images.** For each matched pair, cancerous tissue is on the left, and normal tissue is on the right. 215.25 m/z is known to be more abundant in normal tissue [2]. Note that some of the cancerous tissues appear to have regions of normal tissue, such as samples B, UH0505_12, C, UH0710_33, and F, UH9905_18.

Figure 4.5 shows single ion images for 215.3 m/z, which is an ion known to be more abundant in normal tissue, and Figure 4.6 shows single ion images for 885.7 m/z, which is known to be more abundant in cancerous tissue [2]. Some tissues appear to exhibit heterogeneity, such as the abundance of 215.3 m/z along the edge of the cancerous tissue in sample UH0505_12 (Figure 4.5B). We will use this dataset to demonstrate the ability of the proposed framework to perform classification, while selecting spectral features important in distinguishing the disease condition.

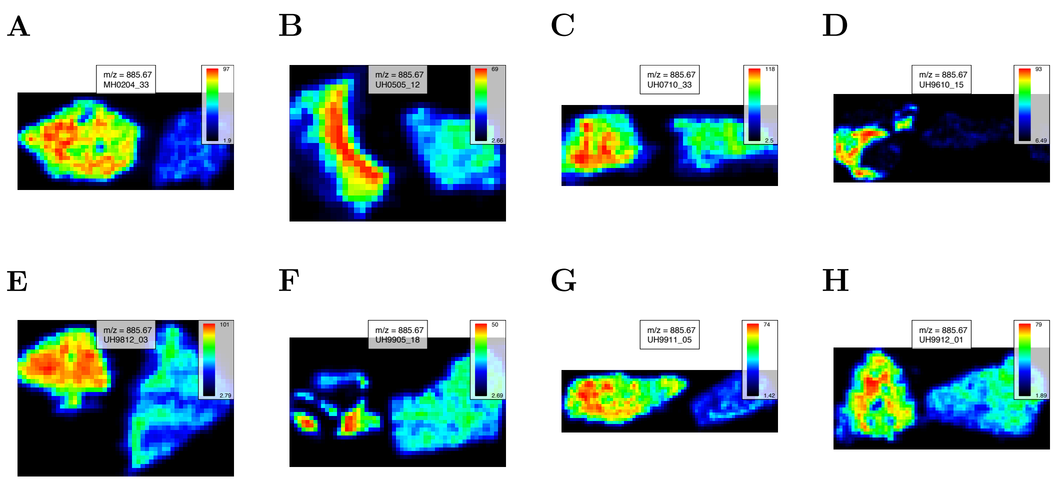


Fig. 4.6. **Human renal cell carcinoma: cancer tissue single ion images.** For each matched pair, cancerous tissue is on the left, and normal tissue is on the right. 885.67 m/z is known to be more abundant in cancerous tissue [2]. Note that some of the normal tissues appear to have regions of cancerous tissue, such as the left edge of sample E, UH9812_03.

4.2 Evaluation for Spatial Shrunken Centroids

4.2.1 Spatial Probabilistic Modeling Improves the Quality of Segmentation over Per-Pixel Segmentation

Spatial segmentations for the pig fetus cross-section dataset are illustrated in Figure 4.7, which compares results from existing segmentation methods with the proposed segmentation method. In Figure 4.7A, k-means clustering was applied to the peak-picked spectra, resulting in a noisy segmentation. The heart is not assigned to a unique segment. Figure 4.7B shows k-means clustering applied to the first five principal components of the peak-picked spectra, which also results in a noisy segmentation, again without the heart represented as a unique segment. Figure 4.7C and Figure 4.7D show the spatially-aware clustering and spatially-aware structurally-adaptive clustering of Alexandrov and Kobarg [7], which both result in cleaner segmentations with clearer edges between segments. The heart is assigned to a unique segment in both segmentations, as well as the brain and liver. All of the methods above require a predetermined number of segments, which was set to 6, based on the procedure described in Section 4.2.2. Figure 4.7E and Figure 4.7F show the proposed spatial shrunken centroids segmentation method with SA and SASA distances, which produce clean segmentations comparable to those in Figure 4.7C and Figure 4.7D. The number of segments for these methods was initialized to 20, and resulted in 6 segments in the final segmentations, as described in Section 4.2.2. In addition, the segmentations in Figure 4.7E and Figure 4.7F are more similar to each other than those in Figure 4.7C and Figure 4.7D, suggesting that the proposed spatial shrunken centroids method produces more consistent results across different types of spatial smoothing.

Spatial segmentations of the cardinal painting are illustrated in Figure 4.8, which demonstrates the performance of existing and proposed methods compared to the ground truth.. In Figure 4.8A, k-means clustering was applied to the peak-picked spectra, resulting in a noisy segmentation. The face (black feathers) are not represented as a unique segment. Figure 4.8B shows k-means clustering applied to the first five principal components of the peak-picked spectra, which also results in a noisy segmentation, but with all parts of the painting represented as segments. Figure 4.8C and Figure 4.8D show the spatially-aware clustering and spatially-aware structurally-adaptive clustering of Alexandrov and Kobarg [7], which both result in cleaner segmentations with clearer edges between segments. The methods above, which require a predetermined number of segments, were set to 8 segments. Figure 4.8E and Figure 4.8F show the proposed spatial shrunken centroids segmentation method with SA and SASA distances, which produce clean segmentations comparable for Figure 4.8C and Figure 4.8D. The proposed method was initialized to 10 segments, resulting in 8 segments in the final segmentations.

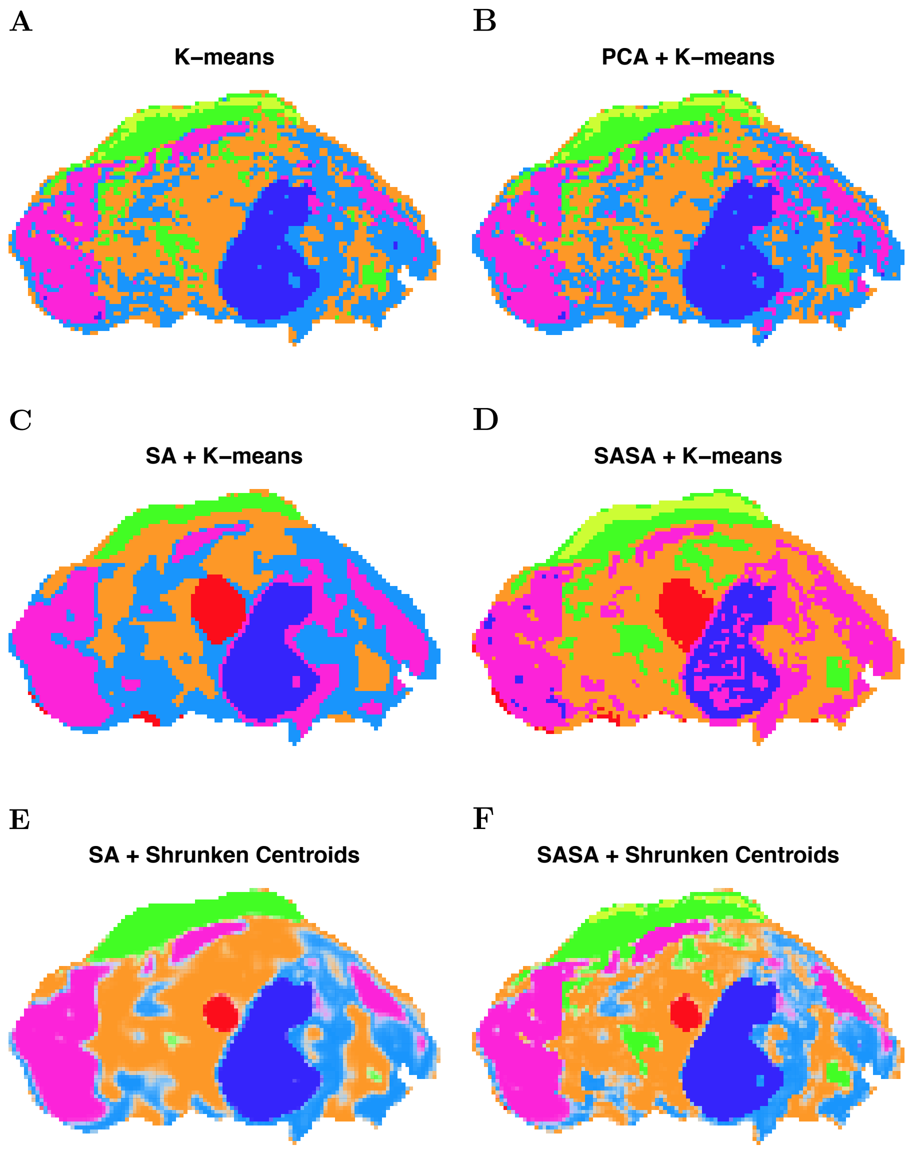


Fig. 4.7. **Pig fetus cross-section: segmentation comparison.** *A*, K-means clustering applied to the peak-picked spectra. *B*, K-means clustering applied to the first five principal components of the peak-picked spectra. *C*, Spatially-aware (SA) clustering. *D*, Spatially-aware structurally-adaptive (SASA) clustering. *E*, Spatial shrunken centroids with SA distance. *F*, Spatial shrunken centroids with SASA distance.

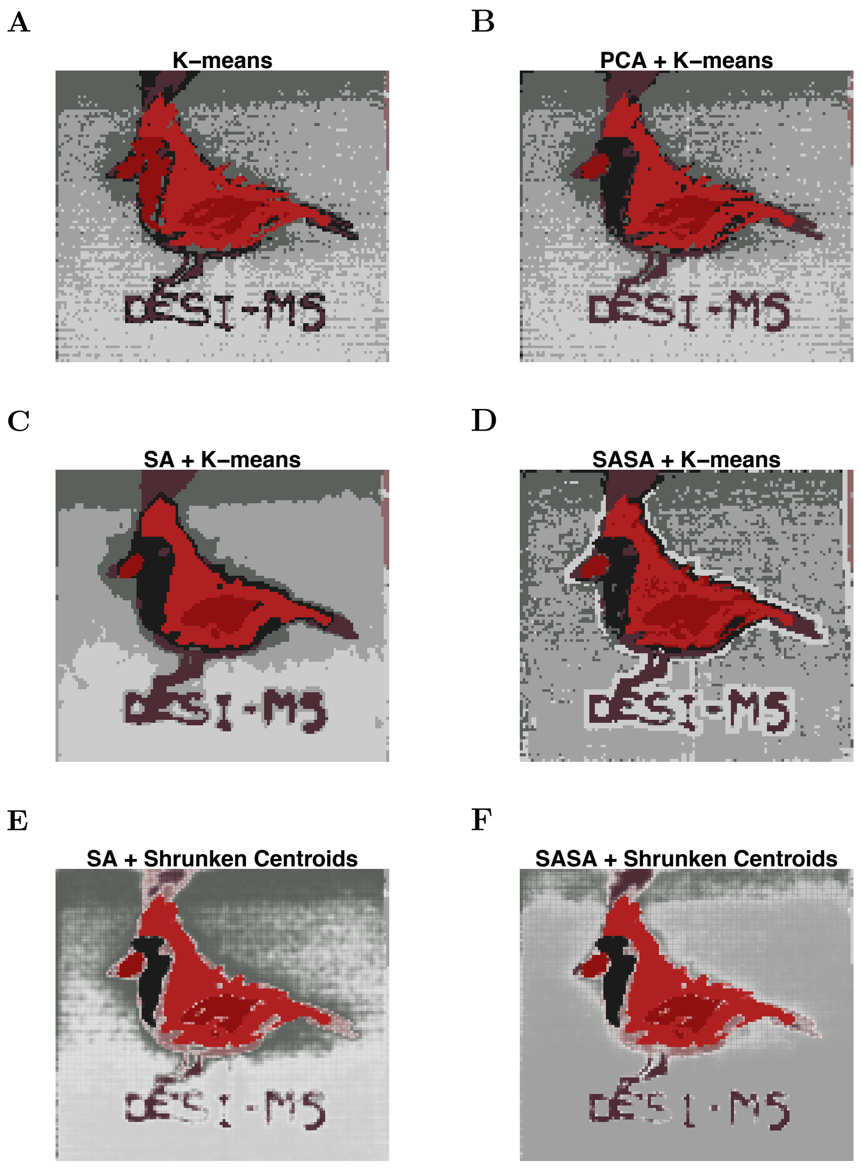


Fig. 4.8. **Cardinal painting: segmentation comparison.** *A*, K-means clustering applied to the peak-picked spectra. *B*, K-means clustering applied to the first five principal components of the peak-picked spectra. *C*, Spatially-aware (SA) clustering. *D*, Spatially-aware structurally-adaptive (SASA) clustering. *E*, Spatial shrunken centroids with SA distance. *F*, Spatial shrunken centroids with SASA.

4.2.2 Statistical Regularization Enables Data-Driven Selection of the Number of Segments for Unsupervised Experiments

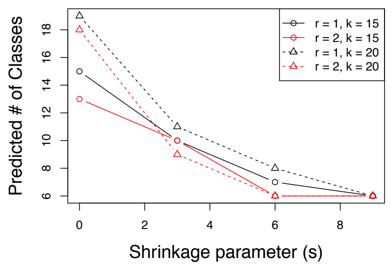
The selection of the number of segments for the pig fetus cross-section dataset is illustrated in Figure 4.9. Figure 4.9A shows the predicted number of segments for increasing shrinkage parameter s for spatial shrunken centroids with the spatially-aware (SA) distance. Figure 4.9B shows the same for spatial shrunken centroids with the spatially-aware structurally-adaptive (SASA) distance. The method was initialized for spatial smoothing radii $r = 1$ and $r = 2$, and for starting number of segments $K = 15$ and $K = 20$. The shrinkage parameter s was increased from 0 to 9 in increments of 3.

To identify segmentations with the most appropriate number of segments, we first look for where the predicted number of segments become similar across different numbers of starting segments K. When this happens, only meaningful segments should remain. This occurs around $s = 3$. Next, we look for where the predicted number of segments stabilizes, which should correspond with an "elbow" in the graph, similar to a scree plot. For Figure 4.9A, this occurs at $s = 6$, but for Figure 4.9B, this may occur earlier at Figure 4.9B $s = 3$.

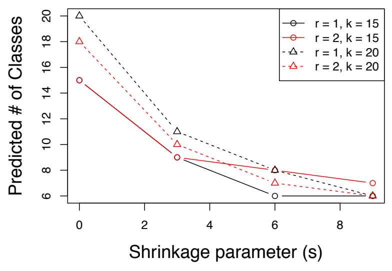
Figure 4.9C–F show the segmentions resulting for increasing shrinkage parameter s for spatial smoothing radius $r = 2$ and starting number of segments $K = 20$.

The selection of the number of segments for the cardinal painting dataset is illustrated in Figure 4.10. Figure 4.10A shows the predicted number of segments for increasing shrinkage parameter s for spatial shrunken centroids with SA distance. Figure 4.10B shows the same for spatial shrunken centroids with SASA distance. The method was initialized for spatial smoothing radii $r = 1$ and $r = 2$, and for starting number of segments $K = 10$ and $K = 15$. The shrinkage parameter s was increased from 0 to 9 in increments of 3. For both versions, the segmentations begin to stabilize around $s = 3$.

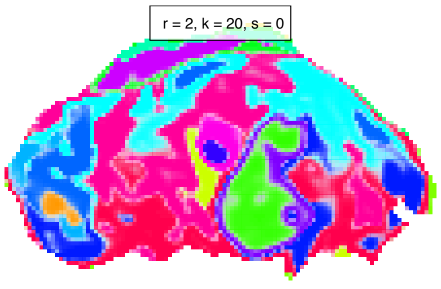
A



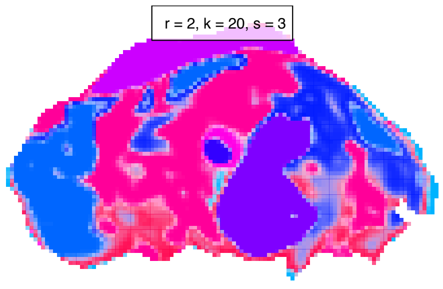
B



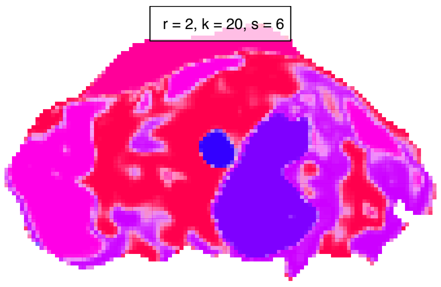
C



D



E



F

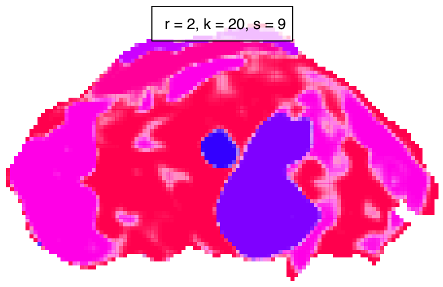


Fig. 4.9. **Pig fetus cross-section: selection of the number of segments.** *A*, Spatially-aware (SA) distance. *B*, Spatially-aware structurally-adaptive (SASA) distance. *C*–*F*, Segmentations using SA distance with smoothing radius of 2 and 20 initial segments, for increasing sparsity parameter s. *C*, $s = 0$, *D*, $s = 3$, *E*, $s = 6$, *F*, $s = 9$.

Figure 4.10C–F show the segmentions resulting for increasing shrinkage parameter s for spatial smoothing radius $r = 2$ and starting number of segments $K = 15$.

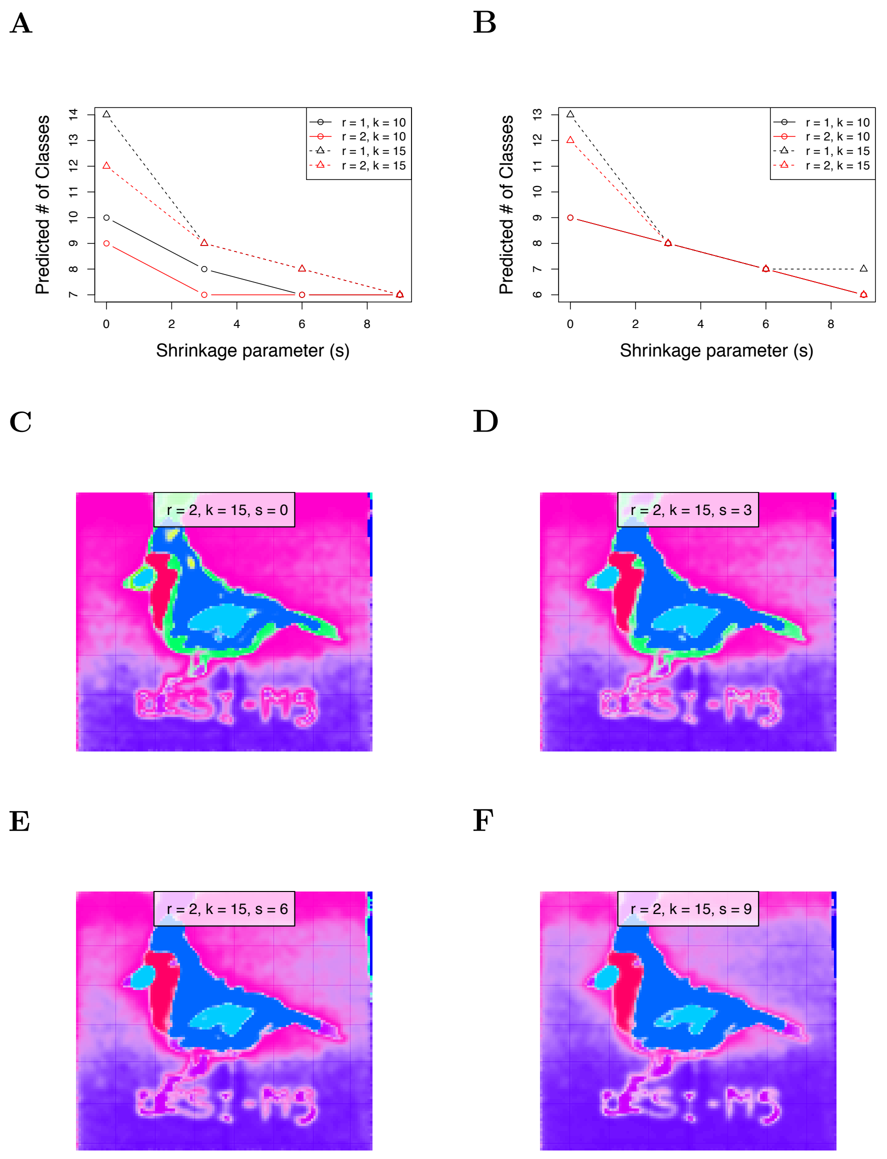


Fig. 4.10. **Cardinal painting: selection of the number of segments.** *A*, Spatially-aware (SA) distance. *B*, Spatially-aware structurally-adaptive (SASA) distance. *C*–*F*, Segmentations using SA distance with smoothing radius of 2 and 20 initial segments, for increasing sparsity parameter s. *C*, $s = 0$, *D*, $s = 3$, *E*, $s = 6$, *F*, $s = 9$.

4.2.3 Feature Selection Aids Interpretation by Automatically Selecting Spectral Features Associated with Differentiating Each Segment from Others

For the pig fetus cross-section segmentation from Figure 4.7E, the selected spectral features using the proposed spatial shrunken centroids segmentation method are shown in Figure 4.11. Feature selection is shown for the brain, heart, and liver segments, along with their corresponding t-statistics and top-ranked single ion images. Note that each unsupervised or supervised segment is characterized by its own reduced subset of informative features. Some features may be found informative for multiple segments, and some features may be found informative for no segment. For the brain segment, 49 spectral features were systematically enriched, and 54 features were systematically absent. For the heart segment, 7 spectral features were systematically enriched, and 1 feature was systematically absent. For the liver segment, 41 spectral features were systematically enriched, and 74 features were systematically absent. Compared to the brain and liver segments, the heart had very few spectral features associated with it.

For the cardinal painting segmentation from Figure 4.8E, the selected spectral features using the proposed spatial shrunken centroids segmentation method are shown in Figure 4.12. Feature selection is shown for the text, body, and wing segments, along with their corresponding t-statistics and top-ranked single ion images. For the text segment, 4 spectral features were systematically enriched, and 27 features were systematically absent. For the body segment, 12 spectral features were systematically enriched, and 36 features were systematically absent. For the wing segment, 13 spectral features were systematically enriched, and 20 features were systematically absent.

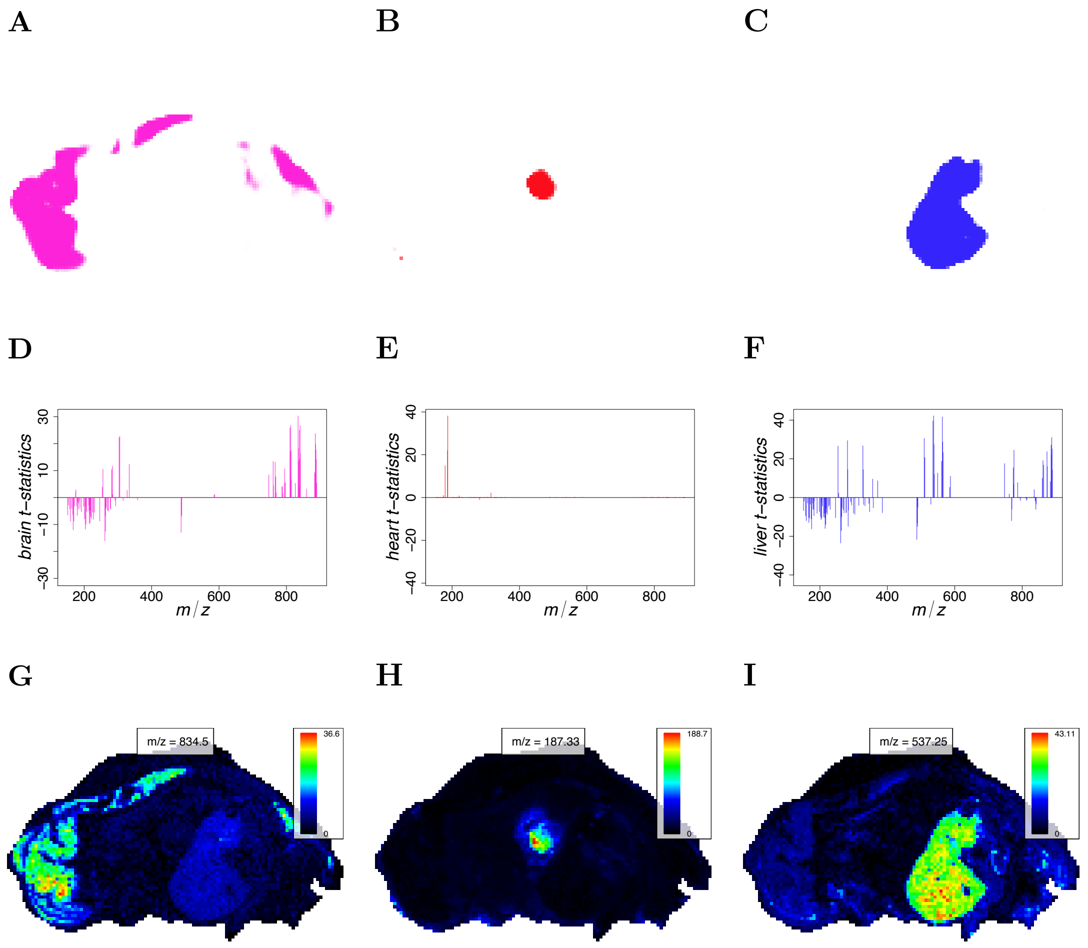


Fig. 4.11. **Pig fetus cross-section: t-statistics and representative single ion images.** *A–C* The predicted segment membership probabilities from spatial shrunken centroids with SA distance. *A*, the brain segment, *B*, the heart segment, and *C*, the liver segment. *D–F* The shrunken t-statistics of the spectral features. *D*, the brain segment, *E*, the heart segment, and *F*, the liver segment. *G–I* The single ion images corresponding with the top-ranked spectral feature by shrunken t-statistic. *G*, the brain segment, *H*, the heart segment, and *I*, the liver segment.

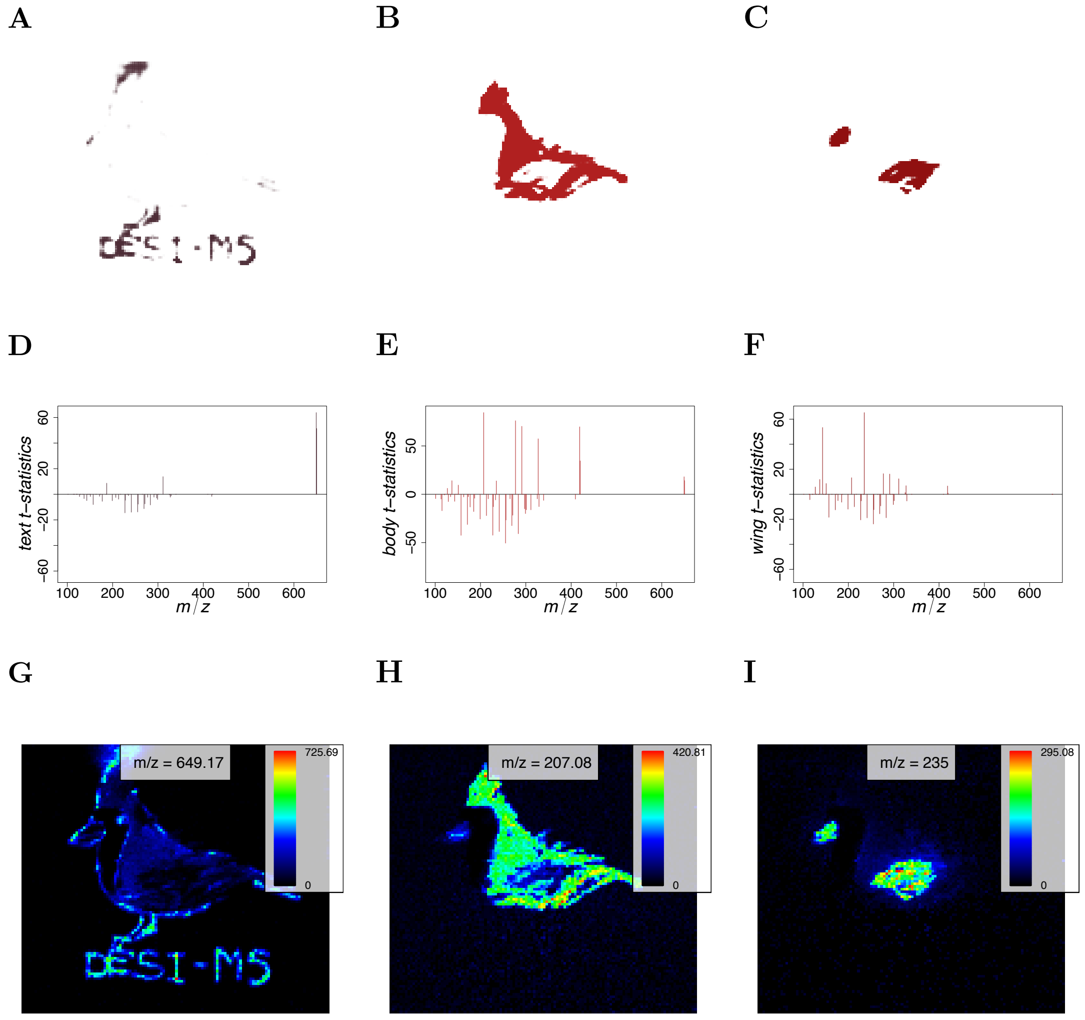


Fig. 4.12. **Cardinal painting: t-statistics and representative single ion images.** *A–C* The predicted segment membership probabilites from spatial shrunken centroids with SA distance. *A*, the text segment, *B*, the body segment, and *C*, the wing segment. *D–F* The shrunken t-statistics of the spectral features. *D*, the text segment, *E*, the body segment, and *F*, the wing segment. *G–I* The single ion images corresponding with the top-ranked spectral features by shrunken t-statistic. *G*, the text segment, *H*, the body segment, and *I*, the wing segment.

4.2.4 Probabilistic Modeling Allows for Characterization and Visual Inspection of Uncertainty in Segment Membership in Unsupervised Experiments

The rodent brain datasets of varying quality were used to evaluate the ability of the proposed method to visually display uncertainty in its resulting segmentations. Because spatial shrunken centroids segmentation results in probabilities of segment membership, using transparency to reflect this probabability creates a straightforward way of visually assessing uncertainty in a segmentation. Segmentations for the 3 rodent brain datasets are compared in Figure 4.13.

Spatial shrunken centroids segmentation was performed for each rodent brain dataset with increasing shrinkage parameter (s), and the "best" segmentations were plotted in Figure 4.13D–F using the criteria described in Section 4.2.2 for selecting an appropriate number of segments. This resulted in 5 segments for both the rat brain (R1) with little noise, 3 segments for the mouse brain (R2) with moderate noise, and 3 segments for the mouse brain (R3) with strong noise. For the strongly noisy mouse brain (R3), there was no clearly appropriate parameter set, as shown in Figure 4.13C. Even for $K = 10$ starting segments, $s = 0$ resulted in only 2 predicted segments, and the predicted number of segments actually increased to 3 temporarily as s was increased before dropping to 2 again. This reflects the lower quality of the information in this brain dataset.

For the sake of comparison, the shrinkage parameter s was further increased past the point of stabilization until the predicted number of segments eventually dropped to only 2 segments. That is, more and more spectral features were excluded from the segmentation until the remaining ones only explained 2 segments. These segmentations are plotted in Figure 4.13G–I. For the rat brain (R1) with little noise, this occured at $s = 25$. For the mouse brain (R2) with moderate noise, this occured at $s = 28$. For the mouse brain (R3) with strong noise, this occured at $s = 0$, reflecting the lesser amount of information in the data.

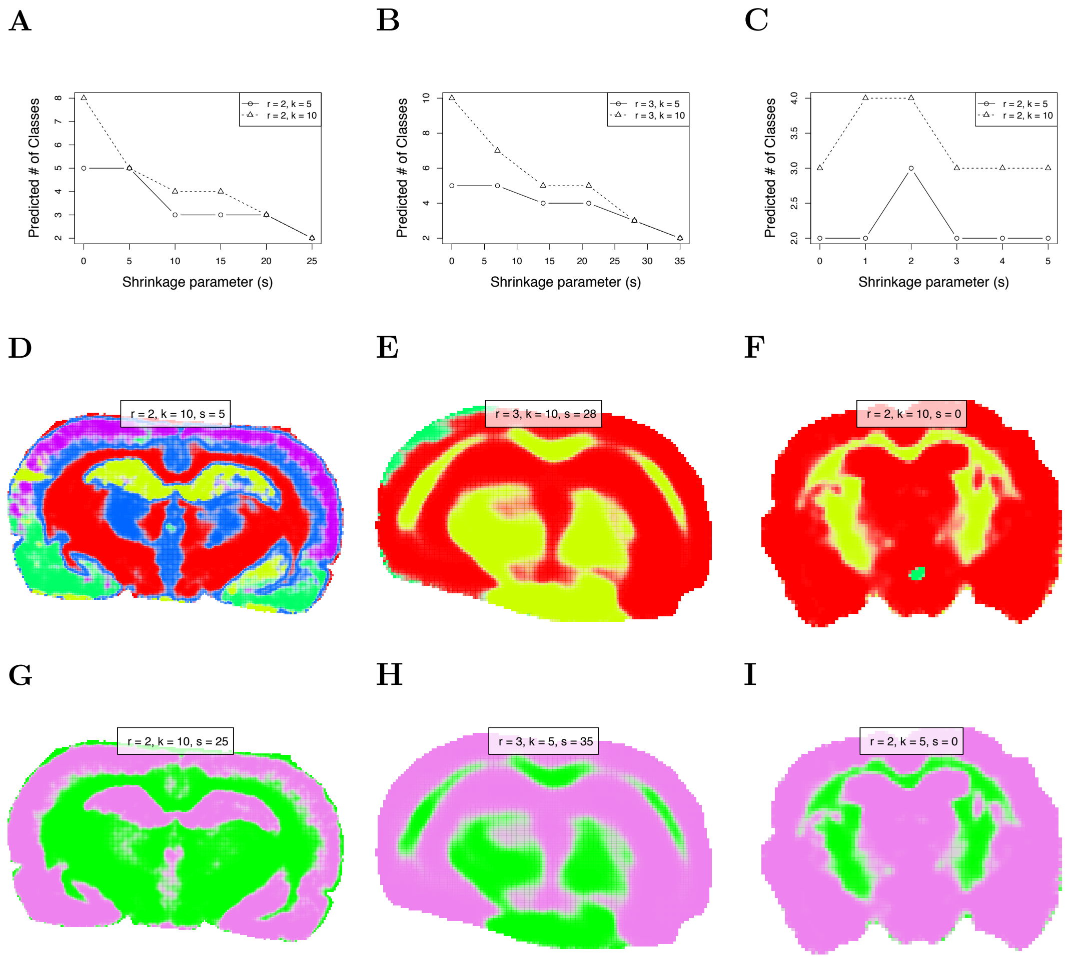


Fig. 4.13. **Comparison of segmentation uncertainty in datasets of differing quality.** *A–C* show the predicted number of segments as the sparsity increases. *A*, rat brain (R1) with little noise. *B*, mouse brain (R2) with moderate noise. *C*, mouse brain (R3) with strong noise. *D–F* show the "best" segmentations selected by choosing the first (least sparse) segmentation after which the predicted number of segments are approximately equal for different initial numbers of segments. *D*, rat brain (R1). *E*, mouse brain (R2). *F*, mouse brain (R3). *G–I* show the segmentations resulting in 2 predicted segments through increasing sparsity. *G*, rat brain (R1). *H*, mouse brain (R2). *H*, mouse brain (R3).

4.2.5 Classification in Supervised Experiments

Classification of the human RCC dataset using the proposed method is illustrated in Figure 4.14 for two of the matched pairs. Eight-fold cross-validation was used to select the shrinkage parameter, as illustrated in Figure 4.15. Spatial shrunken centroids achieves 88.9% cross-validated accuracy, defined as correctly classifying pixels as cancer or normal with respect to the manual annotation of the entire tissue. By comparison, PLS-DA applied to the same dataset achieves 96.8% cross-validated accuracy, and O-PLS-DA achieves 95.4%.

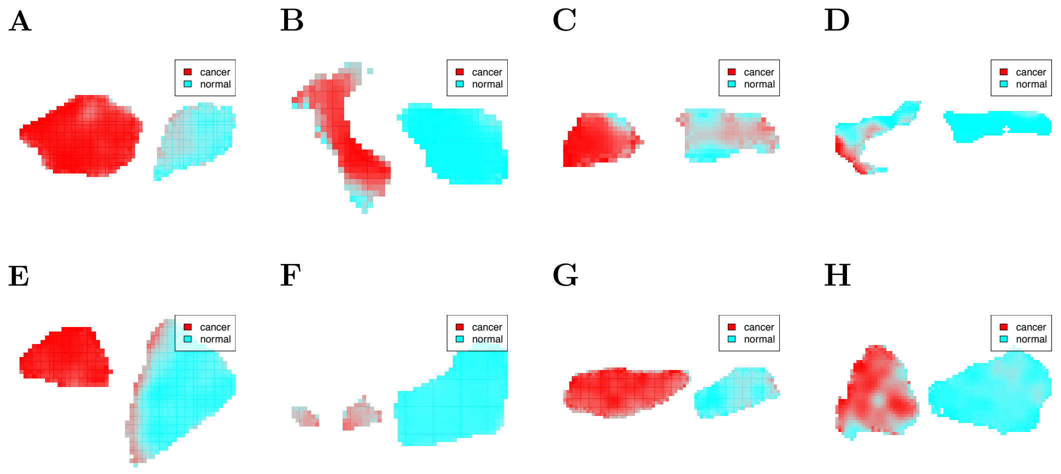


Fig. 4.14. **Human renal cell carcinoma: classification.** For each matched pair, cancerous tissue is on the left, and normal tissue is on the right. Transparency is used to show the predicted probabilities based on spatial shrunken centroids classification. The parameters used were $r = 3, s = 20$, selected by cross-validation, as illustrated in Figure 4.15.

A clear advantage of spatial shrunken centroids for classification is its selection of informative features that differentiate each class, as shown in Figure 4.16C. Unlike PLS-DA and O-PLS-DA, which use all features, making interpretation difficult, spatial shrunken centroids only uses the features that best distinguish each class. Among the selected features, the top ion associated with cancerous tissue was 885.7

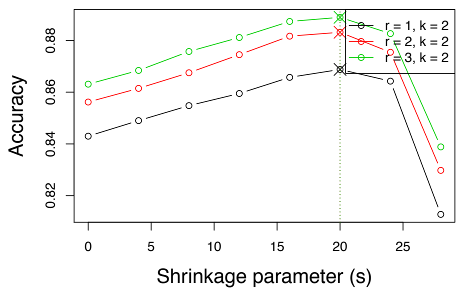


Fig. 4.15. **Human renal cell carcinoma: cross-validation.** The highest cross-validated accuracy rate was for $r = 3, s = 20$ with 88.9% accuracy, defined as correctly classifying a pixel as cancer or normal. Each slide was treated as its own fold in 8-fold cross-validation, i.e., leave-one-sample-out cross-validation.

m/z, which is known to be more abundant in cancer [2]. The top ion associated with normal tissue was 215.3 m/z, which is known to be more abundant in normal tissue [2].

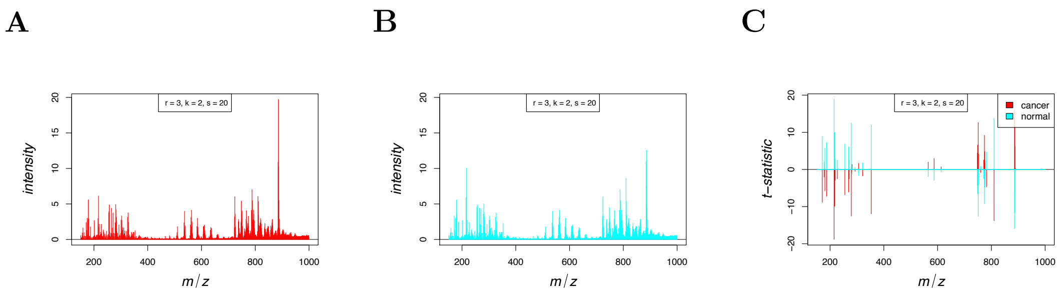


Fig. 4.16. **Human renal cell carcinoma: shrunken centroids and t-statistics.** *A*, the shrunken centroids for cancerous tissue. *B*, the shrunken centroids for normal tissue. *C*, the shrunken t-statistics for normal and cancerous tissue, showing 215 m/z ($t'_{normal,215} = 18.83$) is strongly associated with normal tissue, and 886 m/z ($t'_{cancer,886} = 15.9$) is strongly associated with cancerous tissue.

Another advantage of spatial shrunken centroids is the estimation of probabilities of class membership. Plotting these probabilities with transparency allows visual assessment of the confidence in the prediction. This can help pinpoint heterogeneous regions of the individual tissues, and possible inconsistencies in manual whole-tissue annotations. For example, in Figure 4.14B, the tumor tissue (left) shows an indistinct border of normal tissue along the left side, and in Figure 4.14E, the normal tissue (right) shows an indistinct border of tumor tissue along the left side. These borders are defined by ions known to be associated with cancer and normal tissue [2] (Figure 4.5 and Figure 4.6). Therefore, the manual annotation may be imprecise, and PLS-DA and O-PLS-DA may be overfitting.

4.3 Discussion of Spatial Shrunken Centroids

Spatial shrunken centroids is a general statistical framework for both unsupervised segmentation and supervised classification of MS imaging experiments. For unsupervised segmentation, it produces better segmentations than k-means clustering of the mass spectra, or k-means clustering of their principal components. It outputs probabilities of segment membership, and therefore helps characterize and visualize the uncertainty in the segmentation. It automatically selects the total number of segments, as well as subsets of informative features that define each segment to provide more interpretable results. For supervised classification spatial shrunken centroids achieves similar accuracy as compared to commonly used methods such as PLS-DA and O-PLS-DA. However, similarly to the unsupervised segmentation, it characterizes and visualizes the uncertainty of segment membership, and subsets of informative features that define each class.

Spatial shrunken centroids is designed to work with data obtained after signal processing. It takes as input a set of previously detected, quantified, aligned and normalized features, and is not designed for detecting such features from the raw data anew. Also, spatial shrunken centroids does not require a previous identification on the underlying analytes. The approach only aims at interpreting the quantitative information in the spectra, and this can be done with or without the knowledge of the analyte identity. However, spatial shrunken centroids can potentially enhance the process of identification. For example, the informative subsets of features selected in each segment or class can reduce the possible search space of analytes that we would like to identify.

This framework is implemented in the open-source R package *Cardinal* [8]. We hope that the flexibility, versatility, and efficiency of the method will make it a useful tool for biological and clinical investigations.

5. CARDINAL: OPEN-SOURCE SOFTWARE FOR ANALYSIS OF MS IMAGING EXPERIMENTS

5.1 Overview of *Cardinal*

Cardinal is a free and open-source R package for processing, visualization, and statistical analysis of MS imaging experiments of biological samples such as tissues. It differs from existing software in its focus on statistical analysis and experiments. The companion data package *CardinalWorkflows* includes the pig fetus, cardinal painting, and human RCC datasets described in Chapter 4.

Cardinal contributes statistical methods and statistical computing infrastructure, with a focus on efficiency and scalability.

5.1.1 Applicability and Requirements

Cardinal is applicable to experiments aiming at segmentation and classification of a single tissue, or multiple tissues collected across biological subjects. It is applicable to both DESI and MALDI workflows, and for analyzing either intact or in-situ digested proteins and lipids. *Cardinal* has been tested on raw MS1 spectra from Thermo LTQ linear ion trap, ABSciex TOF/TOF, and Bruker Autoflex MALDI-TOF instruments with resolving powers ranging from 1,000 to 22,000. *Cardinal* is compatible with Windows, Mac and Linux operating systems. The size of the input dataset must be such that it can be loaded entirely into computer memory. *Cardinal* runs optimally when the available memory is twice the size of the dataset.

5.1.2 Data Import, Processing and Visualization

Cardinal supports input data in the imzML format [9], and the `Analyze7.5` format. Free converters to imzML are available for most other formats at `www.imzml.org`, and the converted imzML input data can be read into *Cardinal* .

Cardinal implements a complete set of common spectral processing methods [12], including normalization (e.g., using total ion current), baseline correction (e.g., using median interpolation), peak detection (e.g., using LIMPIC [14]), and peak alignment (e.g., using mean spectrum).

Cardinal visualizes mass spectra, molecular ion images, and results of the statistical analyses. The images are optimized with contrast enhancement and smoothing. The plots can be conditioned on experimental metadata (such as the type of the tissue), and viewed separately using a grid layout with multiple conditions, or jointly in a superposition.

5.1.3 Functionalities for Statistical Analysis

For unsupervised segmentation, *Cardinal* implements several existing methods, e.g. *principle component analysis* (PCA), and *spatially-aware* (SA) and *spatially-aware structurally-adaptive* (SASA) clustering [7]. *Cardinal* also implements the novel method *spatial shrunken centroids*, discussed in Section 3 and Section 4, for model-based unsupervised image segmentation.

For supervised classification, *Cardinal* implements *partial least squares discriminant analysis* (PLS-DA) and *orthogonal projections to latent structures discriminant analysis* (OPLS-DA) [2, 3]. *Cardinal* also implements the supervised version of *spatial shrunken centroids* for model-based image classification, which utilizes the same principles as the model-based image segmentation but works in a supervised manner. For all the methods, *Cardinal* automates the estimation of classification error rate by (cross-)validation.

5.1.4 Implementation and Performance

Cardinal employs efficient data structures to store the data and the metadata, and optimized methods for data manipulation. As a result, *Cardinal* can be used with any dataset that fits in the computer memory. For example, for a dataset with 28,016 pixels that was 2.2 GB before peak picking, and for which the processed version was 63.7 MB after the peak picking, computation of the first 20 principal components took 86.9 sec on the raw data and 4.3 sec on the picked peaks on a MacBook Pro with a 2.6 GHz Intel Core i7 and 16 GB memory. Segmentation with spatial shrunken centroids on the picked peaks took 241 sec (shortest) to 827 sec (longest), depending on the initial values of regularization parameters and the number of clusters, on the same computer.

Cardinal facilitates the development of new functionalities, and interoperability with other software. For example, raw mass spectra can be stored as either a R matrix, or as any matrix-like object, such as a sparse matrix. Most of the processing methods utilize an extendable framework `pixelApply`, similar to the `apply` family of methods in R. The `ResultSet` data structure allows the developers to store the results of any analyses, and directly access the *Cardinal* 's plotting capabilities. *Cardinal* also has functions for simulating mass spectra, to assist method testing. It is publicly available at part of the Bioconductor at `www.bioconductor.org`.

5.2 Design and Implementation of *Cardinal*

Cardinal is designed with two primary purposes in mind: (1) to provide an environment for experimentalists for the handling, pre-processing, analysis, and visualization of mass spectrometry-based imaging experiments, and (2) to provide an infrastructure for computationalists for the development of new computational methods for mass spectrometry-based imaging experiments.

Although MS imaging has attracted the interest of many statisticians and computer scientists, and a number of algorithms have been designed specifically for such experiments, most of these methods remain unavailable to experimentalists, because they are often either proprietary, or difficult for non-experts use. Additionally, the complexity of MS imaging creates a significant barrier to entry for developers. *Cardinal* aims to remove this hurdle, by providing R developers with an accessible way to handle MS imaging data.

5.2.1 S4 Classes

Cardinal extensively uses R's object-oriented S4 class system to handle data and metadata from MS imaging experiments. This section describes the classes used by *Cardinal* .

The `iSet` object is the foundational data structure of *Cardinal* .

- similar to `eSet` in *Biobase* and `pSet` in *MSnbase* (from Bioconductor).
- coordinates high-throughput imaging data, feature data, pixel data, and experimental metadata.
- provides an interface for manipulating data from imaging experiments.

Just as `eSet` from *Biobase* coordinates gene expression data and `pSet` from *MSnbase* coordinates proteomics data, `iSet` coordinates imaging data. It is a virtual class, so it is used only through its subclasses.

`MSImageSet` is a subclass of `iSet`, and is the primary data structure used in *Cardinal* . It is designed to coordinate data from mass spectrometry-based imaging experiments. It contains mass spectra (or mass spectral peaks), feature data (including m/z values), pixel data (including pixel coordinates and phenotype data), and other metadata. When a raw MS image data file is read into *Cardinal* , it is turned into an `MSImageSet`, which can then be used with *Cardinal* 's methods for pre-processing, analysis, and visualization.

`MSImageData` is the class responsible for coordinating the mass spectra themselves, and reconstructing them into images when necessary. Every `MSImageSet` has an `imageData` slot containing an `MSImageData` object. It is similar to the `assayData` slot in *Biobase*, in that it uses an `environment` to store large high-throughput data more efficiently in memory, without R's usual copy-on-edit behavior.

`IAnnotatedDataFrame` extends the *Biobase* `AnnotatedDataFrame` class by making a distinction between *pixels* and *samples*. An `IAnnotatedDataFrame` tracks pixel data, where each row corresponds to a single pixel, and each column corresponds to some measured variable (such as phenotype). An `MSImageSet` may contain multiple samples, where each sample is a single image, and possibly thousands of pixels corresponding to each sample.

`ResultSet` is a class for containing results of analyses performed on `iSet` objects. A single `ResultSet` object may contain results for multiple parameter sets. Using a `ResultSet` provides users and developers with a standard way of viewing and plotting the results of analyses.

Together, these classes (along with a few others) provide a useful way of accessing and manipulating MS imaging data while keeping track of important experimental metadata.

`iSet`: High-Throughput Imaging Experiments

Inspired by `eSet` in *Biobase* and `pSet` in *MSnbase*, the virtual class `iSet` provides the foundation for other classes in *Cardinal* . It is a generic class for the storage of imaging data and experimental metadata.

Structure:

- `imageData`: high-throughput image data
- `pixelData`: pixel covariates (coordinates, sample, phenotype, etc.)
- `featureData`: feature covariates (m/z, protein annotation, etc.)
- `experimentData`: experiment description
- `protocolData`: sample protocol

Of particular note is the `imageData` slot for the storing of high-throughput image data, which will be discussed further in Section 5.2.1, and the `pixelData` slot, which will be discussed further in Section 5.2.1.

`SImageSet`: Pixel-Sparse Imaging Experiments

`SImageSet` extends `iSet` without extending its internal structure. `SImageSet` implements methods assuming that the structure of `imageData` is a (# of features) x (# of pixels) matrix, where each column corresponds to a pixel's feature vector (e.g., a single mass spectrum), and each row corresponds to a vector of flattened image intensities.

`SImageSet` further assumes that there may be a number of missing pixels in the experiment. This is useful for non-rectangular images, and experiments with multiple images of different dimensions.

`MSImageSet`: Mass Spectrometry-based Imaging Experiments

`MSImageSet` extends `SImageSet` with mass spectrometry-specific features, including expecting m/z values to be stored in the `featureData` slot. This is the primary class in *Cardinal* for handling MS imaging experiments. It also adds a slot `processingData` for tracking the what pre-processing has been applied to the dataset.

`ImageData`: High-Throughput Image Data

`iSet` and all of its subclasses have an `imageData` slot for storing the high-throughput image data. This must be an object of class `ImageData` or one of its subclasses.

Similar to the `assayData` slot in `eSet` from *Biobase* and `pSet` from *MSnbase*, `ImageData` uses an `environment` as its `data` slot to store data objects in memory more efficiently, and bypass R's usual copy-on-edit behavior. Because these data elements of `ImageData` may be very large, editing any metadata in an `iSet` object would trigger expensive copying of these large data elements if a usual R `list` were used. Using an `environment` avoids this behavior.

`ImageData` makes no assumptions about the class of objects that make up the elements of its `data` slot, but they must be array-like objects that return a positive-length vector to a call to `dim`. These data elements must also have the same number of dimensions, but they may have different extents.

Structure:

- `data`: high-throughput image data
- `storageMode`: mode of the `data` environment

Similar to `assayData`, the elements of `ImageData` can be stored in three different ways. These are as a *immutableEnvironment*, *lockedEnvironment*, or *environment*.

The modes *lockedEnvironment* and *environment* behave the same as for `assayData` in *Biobase* and *MSnbase*. *Cardinal* introduces *immutableEnvironment*, which is a compromise between the two. When the storage mode is *immutableEnvironment*, only changing the values of the elements of `ImageData` directly will trigger copying, while changing object metadata will not trigger copying.

`SImageData`: Pixel-Sparse Imaging Experiments

While `ImageData` makes very few assumptions about the objects that are the elements of its `data` slot, its subclass `SImageData` expects a very specific structure to its data elements.

`SimageData` expects at least one element named "iData" (accessed by `iData`) which is a (# of features) x (# of pixels) matrix, where each column is a feature vector (i.e., a single mass spectrum) associated with a single pixel, and each row is a vector of flattened image intensities. Additional elements should follow the same structure, with the same dimensions.

Structure:

- `data`: high-throughput image data
- `storageMode`: mode of the `data` environment
- `coord`: `data.frame` of pixel coordinates.
- `positionArray`: `array` mapping coordinates to pixel column indices
- `dim`: dimensions of array elements in `data`
- `dimnames`: dimension names

`SimageData` implements methods for re-constructing images from the rows of flattened image intensities on-the-fly. In addition, it assumes the images may be pixel-sparse. This means data for missing pixels does not need to be stored. Instead, the `positionArray` slot holds an `array` of the same dimension as the *true dimensions* of the imaging dataset, i.e., the maximum of each column of `coord`. For each pixel coordinate from the *true image*, the `positionArray` stores the index of the column for which the associated feature vector is stored in the matrix elements of `data`.

This allows transforming the image (e.g., changing the pixel coordinates such as transposing the image, rotating it, etc.) without editing (and thereby triggering R to make a copy of) the (possibly very large) data matrix elements in `data`. This also means that it doesn't matter what order the pixels' feature vectors (e.g., mass spectra) are stored.

`MSImageData`: Mass Spectrometry Imaging Data

`MSImageData` is a small extension of `SImageData`, which adds methods for accessing additional elements of `data` specific to mass spectrometry. There are an element named "peakData" (accessed by `peakData`) for storing the intensities of peaks, and "mzData" (accessed by `mzData`) for storing the m/z values of peaks. Generally, these elements will only exist after peak-picking has been performed. (They may not exist if the data has been reduced to contain *only* peaks, i.e., if the “iData” element consists of peaks rather than full mass spectra.)

The “peakData” and “mzData” elements (when they exist) are usually objects of class `Hashmat`.

`Hashmat`: Compressed-Sparse Column Matrices

The `Hashmat` class is a compressed-sparse column matrix implementation designed to store mass spectral peaks efficiently alongside full spectra, and allow dynamic filtering and re-alignment of peaks without losing data.

Structure:

- `data`: sparse data matrix elements
- `keys`: identifiers of non-zero elements
- `dim`: dimensions of (full) matrix
- `dimnames`: dimension names

In a `Hashmat` object, the `data` slot is a `list` where each element is a column of the sparse matrix, represented by a named `numeric` vector. The `keys` slot is a `character` vector. The columns of the dense matrix are reconstructing by indexing each of the named vectors in `data` by the `keys`. This means that a `Hashmat` can store matrix elements that are selectively zero or non-zero depending on the keys.

In the context of mass spectral peak-picking, this means that each sparse column is a vector of mass spectral peaks. Peaks can be filtered (e.g., removing low-intensity peaks) or aligned (e.g., to the mean spectrum) loss-lessly, by changing the `keys`. Filtering peaks simply means deleting a key, while peak alignment simply means re-arranging the keys. Additionally, the dimension of the dense matrix will be the same as the full mass spectra, while requiring very little additional storage.

`IAnnotatedDataFrame`: Pixel Metadata for Imaging Experiments

`IAnnotatedDataFrame` is extension of `AnnotatedDataFrame` from *Biobase*. It serves as the `pixelData` slot for `iSet` and its subclasses. In an `AnnotatedDataFrame`, each row corresponds to a sample. However, in an `IAnnotatedDataFrame`, each row instead corresponds to a pixel.

In an imaging experiment, each image is a sample, and a single image is composed of many pixels. Therefore, `IAnnotatedDataFrame` may have very many pixels, but have very few (or even just a single) sample.

An `IAnnotatedDataFrame` must have a column named "sample", which is a `factor`, and gives the sample to which each pixel belongs.

For an `IAnnotatedDataFrame`, `pixelNames` retrieves the row names, while `sampleNames` retrieves the levels of the "sample" column.

In addition, `varMetadata` must have a column named "labelType", which is a `factor`, and takes on the values "pheno", "sample", or "dim". If a variable is "dim", then it describes pixel coordinates; if a variable is "sample", then the variable is the "sample" column *and it is not currently acting as a pixel coordinate*; if a variable is "pheno", then it is describing phenotype.

Note that the "sample" column may sometimes act as a pixel coordinate, in which case its "labelType" will be "dim", while all other times its "labelType" will be "sample".

`MIAPE-Imaging`: Minimum Information about a Proteomics Experiment for MS imaging

For `MSImageSet` objects, the `experimentData` slot must be an object of class `MIAPE-Imaging`. That is the Minimum Information About a Protemics Experiment for Imaging. Most of its unique slots are based on the imzML specification [9].

`MSImageProcess`: Mass Spectral Pre-processing Information

`MSImageSet` objects also have a `processingData` slot, which must be an object of class `MSImageProcess`. This gives information about the pre-processing steps that have been applied to the dataset. All of the standard pre-processing methods in *Cardinal* will fill in `processingData` with the appropriate processing type automatically.

`ResultSet`: Analysis Results for Imaging Experiments

`ResultSet` is a subclass of `iSet`, and is used to store the results of analyses applied to `iSet` and `iSet`-derived objects.

In addition to the usual `iSet` slots, a `ResultSet` also has a `resultData` slot, which is a `list` used to store results, and a `modelData` slot, which describes the parameters of the fitted model. The `ResultSet` class assumes that multiple models may be fit (i.e., multiple parameter sets over a grid search). Therefore, each element of the `resultData` `list` should be another `list` containing the results for a single model, and each row of `modelData` should describe the parameters for that one model.

5.2.2 Visualization

Cardinal implements a powerful graphics interface inspired by the *lattice* graphics package's implementation of trellis plots [27]. Both *Cardinal* 's `image` method for plotting of images and its `plot` method for plotting of mass spectra take formulas that allow conditioning on grouping variables, as well as other *lattice*-inspired arguments that enable simple formulations for creating complex plots.

To demonstrate a subset of *Cardinal* 's plotting functionality, we will use the cardinal painting dataset described in Section 4.1 and shown Figure 4.2. We load the dataset from the companion data package *CardinalWorkflows* , and request the top 9 features selected by spatial shrunken centroids (by greatest t-statistic) in the segmentation shown in Figure 4.8E.

```
> library(CardinalWorkflows)
> data(cardinal, cardinal_analyses)
> top <- topLabels(cardinal.sscg, model=list(r=1, k=10, s=3), n=9)
```

Now we plot their molecular ion images with contrast enhancement via histogram equalization, and normalizing their intensities so that each ion image has the same intensity range.

```
> image(cardinal, mz=top$mz, plusminus=0.5, normalize.image="linear",
+     contrast.enhance="histogram", layout=c(3,3))
```

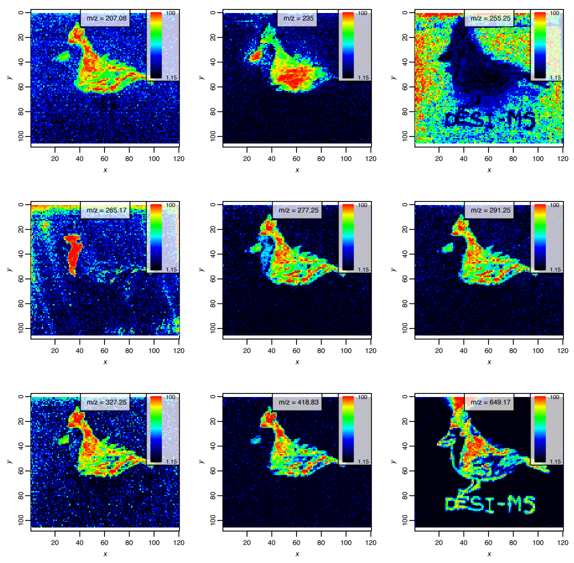


Fig. 5.1. **Trellis display of molecular ion images using** *Cardinal* **.** Top-ranked ion images for the cardinal painting dataset by t-statistic.

Using *Cardinal* 's plotting methods, we can recreate the painting by overlaying specific ion images. Based on the ion images in Figure 5.1, we choose m/z 207.08 to represent the cardinal's body in red, m/z 235 to represent the cardinal's wing in dark red, m/z 255.25 to represent the gray background, m/z 265.17 to represent the cardinal's face in black, and m/z 649.17 to represent the text in brown. It is straightfoward to overlay the ion images by setting `superpose=TRUE`.

```
> image(cardinal, mz=c(207.08, 235, 255.25, 265.17, 649.17),
+     plusminus=0.5,
+     normalize.image="linear",
+     contrast.enhance="histogram",
+     col=c("red", "darkred", "gray", "black", "brown"),
+     superpose=TRUE)
```

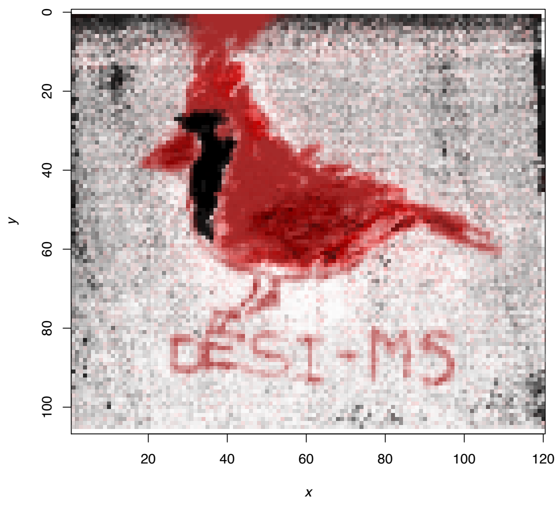


Fig. 5.2. **Overlay of molecular ion images using** *Cardinal* **.** Recreation of the cardinal painting using overlaid ion images.

5.2.3 `pixelApply` and `featureApply`

The `apply` family of functions are a powerful feature of statistical computing in R. The `apply` function applies a function over margins of an array, while `sapply` applies a function over every element of a vector-like object. The function `tapply` applies a function over a "ragged" array, so that the function is applied over groups of values given by levels of another variable (usually a factor). *Cardinal* provides the methods `pixelApply` and `featureApply` for `apply`-like functionality that combine traits of each of these, tailored specifically for mass spectrometry imaging datasets. Their most important parameters are:

- `.object` the `MSImageSet` with the experimental data
- `.fun` the function to be applied over the dataset
- `.pixel` which pixels the function should be applied over
- `.feature` which features the function should be applied over
- `.pixel.groups` which pixels should be grouped together and the function applied to them seperately
- `.feature.groups` which features should be grouped together and the function applied to them seperately

In fact, many of the spectral processing methods *Cardinal* provides are internally implement through the use of `pixelApply` and `featureApply`.

Below we show a toy example of how TIC normalization and standardization of samples could both be implemented very easily using `pixelApply` and `featureApply`. Note that these are simplifications for the sake of demonstration, and do not reflect all of the concerns that must be taken into account when doing normalization of mass spectra or standardization of samples on real data.

```
> standardize <- function(x) x / sum(x)
> normalize.tic <- function(data) {
+     pixelApply(data, .fun=standardize)
+ }
> standardize.samples <- function(data) {
+     featureApply(data, .fun=standardize, .pixel.groups=sample)
+ }
```

The definition of `normalize.tic` is straightforward and applies TIC normalization to the mass spectrum of each pixel. The definition of `standardize.samples` shows `tapply`-like functionality by applying standardization separately to each sample by specifying `.pixel.groups=sample`.

5.2.4 Simulation of Test Datasets

Cardinal provides functions for the simulation of mass spectra and mass spectrometry imaging datasets. This is particularly important for statisticians and other developers for testing newly developed methodology for analyzing mass spectrometry imaging experiments.

Simulation of Spectra

The `generateSpectrum` function can be used to simulate mass spectra. Its parameters can be tuned to simulate different kinds of mass spectra from different kinds of machines, and different protein and peptide patterns.

One spectrum with m/z range from 1001 to 20000, 50 randomly selected peaks, baseline 3000, and m/z resolution 100 is generated below and plotted in Figure 5.3A.

```
> set.seed(1)
> s1 <- generateSpectrum(1, range=c(1001, 20000), centers=runif(50,
+     1001, 20000), baseline=2000, resolution=100, step=3.3)
> plot(x ~ t, data=s1, type="l", xlab="m/z", ylab="Intensity")
```

An example with fewer peaks, larger baseline, and lower resolution (Figure 5.3B):

```
> set.seed(2)
> s2 <- generateSpectrum(1, range=c(1001, 20000), centers=runif(20,
+     1001, 20000), baseline=3000, resolution=50, step=3.3)
> plot(x ~ t, data=s2, type="l", xlab="m/z", ylab="Intensity")
```

Above we simulated MALDI-like spectra. We can also simulate DESI-like spectra, shown in Figure 5.4.

```
> set.seed(3)
> s3 <- generateSpectrum(1, range=c(101, 1000), centers=runif(25,
+     101, 1000), baseline=0, resolution=250, noise=0.1, step=1.2)
> plot(x ~ t, data=s3, type="l", xlab="m/z", ylab="Intensity")
```

```
> set.seed(4)
> s4 <- generateSpectrum(1, range=c(101, 1000), centers=runif(100,
+     101, 1000), baseline=0, resolution=500, noise=0.2, step=1.2)
> plot(x ~ t, data=s4, type="l", xlab="m/z", ylab="Intensity")
```

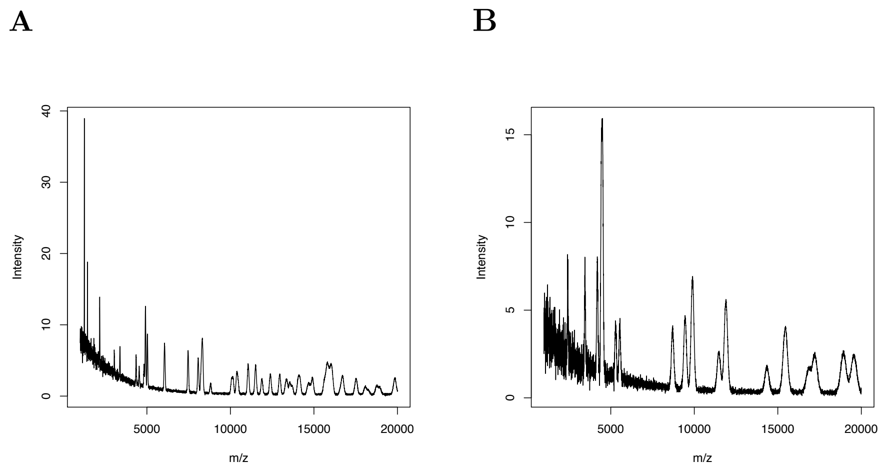


Fig. 5.3. **MALDI-like simulated spectra.**

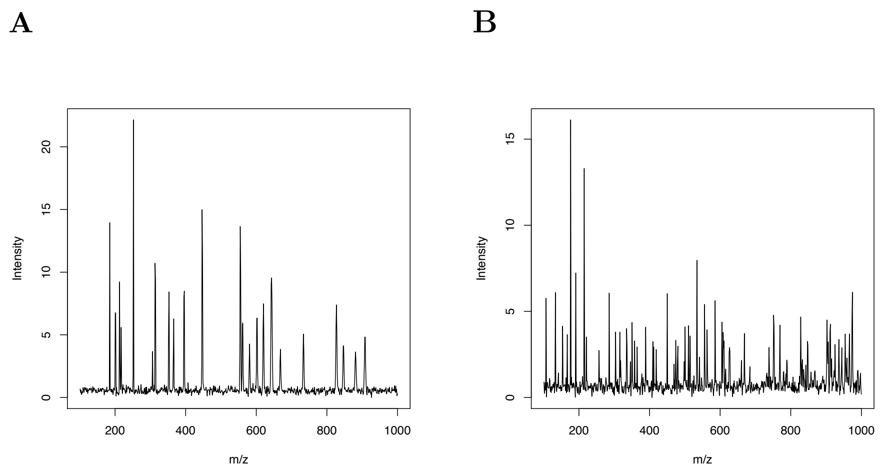


Fig. 5.4. **DESI-like simulated spectra.**

Simulation of Images

The `generateImage` function can be used to simulate mass spectral images. This is a simple wrapper for `generateSpectra` that will generate unique spectral patterns based on a spatial pattern. The generated mass spectra will have a unique peak associated with each region. The pattern must have discrete regions, most easily given in the form of an integer matrix. We use a matrix in the pattern of a cardinal.

```
> data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
+     NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
+     1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
+     1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
+     1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA),
+     nrow=9, ncol=9)
```

As seen in Figure 5.5, we can plot the ground truth image directly.

```
> image(data[,ncol(data):1], col=c("black", "red"))
```

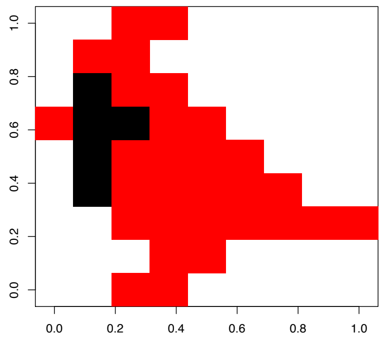


Fig. 5.5. **Ground truth image used to generate the simulated dataset.**

Below, we generate the dataset. To make it easy to visualize, we set up the `range` and `step` size so that the feature indices correspond directly to their values. We create two peaks at m/z 100 and m/z 200, one of which is associated with each region in the image.

```
> set.seed(1)
> img1 <- generateImage(data, range=c(1, 1000), centers=c(100, 200),
+     step=1, as="MSImageSet")
```

Now to confirm the reasonability of our simulated dataset, we plot images corresponding to the two peaks associated with each region in Figure 5.6. (Note that rows in the original matrix correspond to the x-axis in the image and the columns correspond to the y-axis.)

```
> image(img1, mz=100, col.regions=alpha.colors(100, "black"))
> image(img1, mz=200, col.regions=alpha.colors(100, "red"))
```

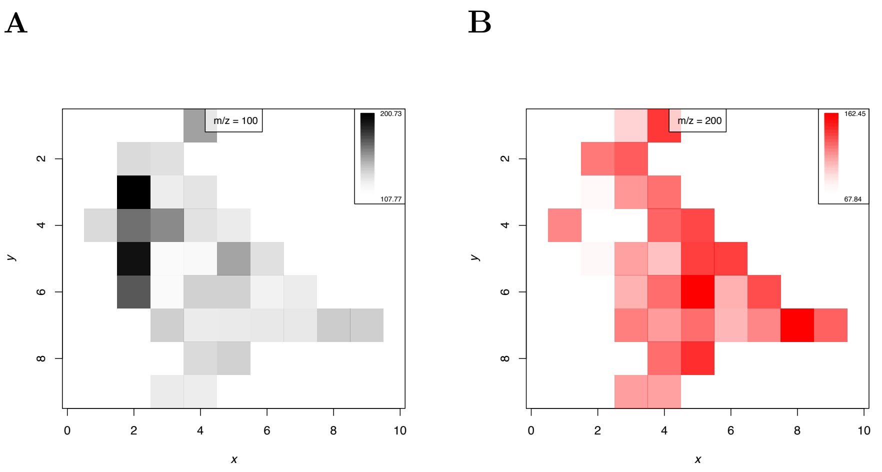


Fig. 5.6. **Generated image from an integer matrix.** *A*, black peak. *B*, red peak.

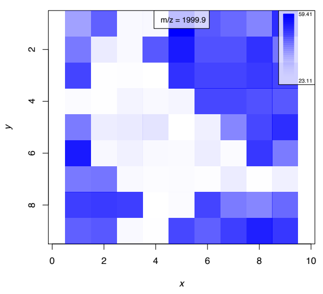
We can generate the same kind of dataset using a `factor` and a `data.frame` of coordinates, as is done in the running example for earlier sections of this vignette.

```
> pattern <- factor(c(0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 0,
+     0, 0, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0, 0, 0, 2, 1, 1, 2,
+     2, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 0, 0, 0, 0, 1, 2, 2,
+     2, 2, 2, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 2,
+     2, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0),
+     levels=c(0,1,2), labels=c("blue", "black", "red"))
> coord <- expand.grid(x=1:9, y=1:9)
> set.seed(2)
> msset <- generateImage(pattern, coord=coord, range=c(1000, 5000),
+     centers=c(2000, 3000, 4000), resolution=100, step=3.3,
+     as="MSImageSet")
```

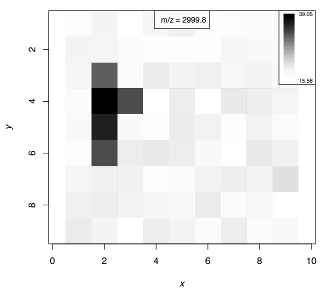
Again, we can plot the images to see that the simulated dataset is the same pattern as before (though the exact intensities will differ, because we have used a different seed for the random number generator), Figure 5.7.

```
> image(msset, mz=2000, col.regions=alpha.colors(100, "blue"))
> image(msset, mz=3000, col.regions=alpha.colors(100, "black"))
> image(msset, mz=4000, col.regions=alpha.colors(100, "red"))
```

A



B



C

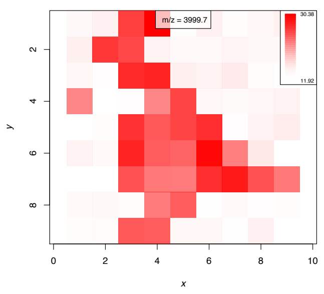


Fig. 5.7. **Generated images from factor and coordinates.** *A*, blue peak. *B*, black peak. *C*, red peak.

Advanced Simulation

The `generateImage` function provides a straightforward method for rapid simulation of many kinds of images to test classification and segmentation models, but suppose we wish to simulate a more complex dataset with spatial correlations. Below we simulate a dataset with two overlapping regions. In each of these regions, the intensity degrades with distance from the center of the region, implining spatial correlation, Figure 5.8.

```
> x1 <- apply(expand.grid(x=1:10, y=1:10), 1,
+     function(z) 1/(1 + ((4-z[[1]])/2)^2 + ((4-z[[2]])/2)^2))
> dim(x1) <- c(10,10)
> image(x1[,ncol(x1):1])
> x2 <- apply(expand.grid(x=1:10, y=1:10), 1,
+     function(z) 1/(1 + ((6-z[[1]])/2)^2 + ((6-z[[2]])/2)^2))
> dim(x2) <- c(10,10)
> image(x2[,ncol(x2):1])
```

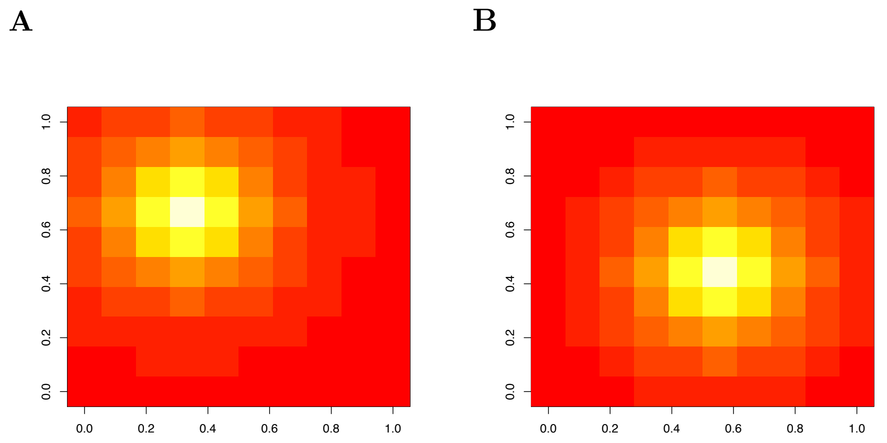


Fig. 5.8. **Ground truth images of a dataset with overlapping regions.** *A*, region 1. *B*, region 2.

We generate the image by using `generateSpectrum` with the calculated mean intensities. We use two peaks for the two regions with nearly overlapping peaks at m/z 500 and m/z 510.

```
> set.seed(1)
> x3 <- mapply(function(z1, z2) generateSpectrum(1, centers=c(500,510),
+     intensities=c(z1, z2), range=c(1, 1000), resolution=100,
+     baseline=0, step=1)$x, as.vector(x1), as.vector(x2))
> img3 <- MSImageSet(x3, coord=expand.grid(x=1:10, y=1:10), mz=1:1000)
```

Below, we plot the ion images for each of the two peaks in Figure 5.9.

```
> image(img3, mz=500, col=intensity.colors(100))
```

```
> image(img3, mz=510, col=intensity.colors(100))
```

Finally, we plot the mass spectrum for a pixel from each region in Figure 5.10.

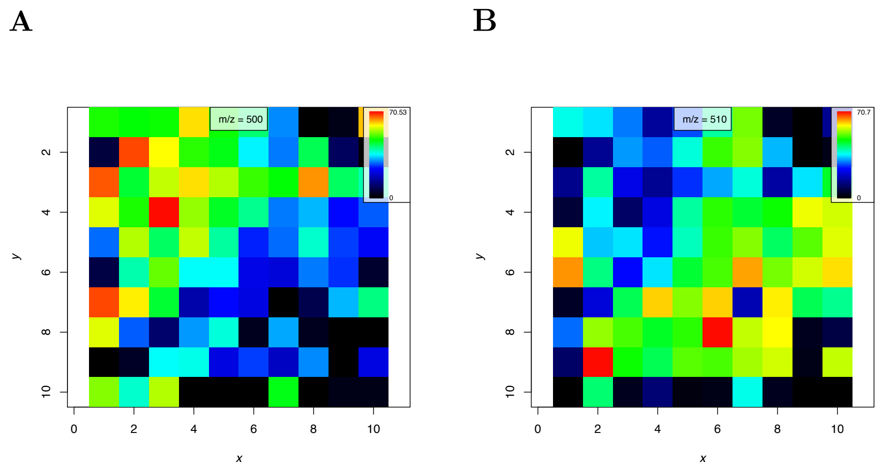


Fig. 5.9. **Simulated images at the two peaks.** *A*, m/z 500. *B*, m/z 510.

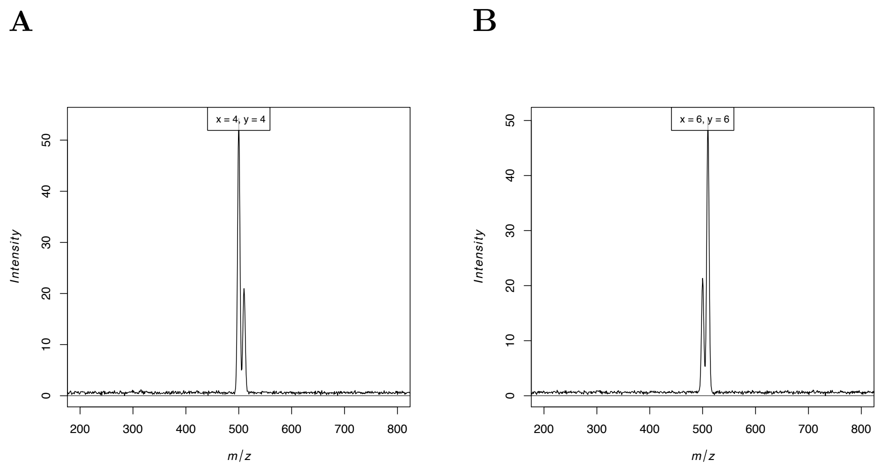


Fig. 5.10. **Simulated mass spectra from the two regions.** *A*, region 1, pixel 34. *B*, region 2, pixel 56.

```
> plot(img3, coord=list(x=4, y=4), type="l", xlim=c(200, 800))

> plot(img3, coord=list(x=6, y=6), type="l", xlim=c(200, 800))
```

By creating spatial correlation patterns and combining them with the `intensities`, `sd`, and `noise` arguments in `generateSpectrum`, it is possible to simulate more complex mass spectrometry imaging datasets.

5.3 *Cardinal* Examples

5.3.1 Unsupervised Segmentation Workflow

Here we show how *Cardinal* was used to analyze the pig fetus dataset shown in Figure 4.1 and produce the segmentation results shown in Figure 4.7 from Section 4.1.

```
> library(CardinalWorkflows)
> data(pig206)
> summary(pig206)
```

```
Class: MSImageSet
Features: m/z = 150.08 ... m/z = 1000 (10200 total)
Pixels: x = 72, y = 1 ... x = 83, y = 66 (4959 total)
x: 10 ... 120
y: 1 ... 66
Size in memory: 195.4 Mb
```

Normalization

In order to ensure that the spectra are comparable pixel-to-pixel, normalization is often done as a pre-processing step. As described in Section 2.1.2, a popular choice for normalization in mass spectrometry image analysis TIC normalization.

```
> pig206.norm <- normalize(pig206, method = "tic")
```

Peak Picking and Alignment

For computational efficency, it is necessary to do peak picking prior to analysing the data. As in Alexandrov et al. [26], we peak pick on every 10th mass spectrum, retaining only those peaks that occur in at least 1% of the considered spectra. The selection of peaks in *Cardinal* is done using a comparison of local maxima against noise.

First, we perform peak-picking on ever 10^{th} mass spectrum using the `peakPick` method, looking for peaks with a signal-to-noise ratio (SNR) of at least 6.

```
> pig206.peaklist <- peakPick(pig206.norm, pixel = seq(1, ncol(pig206),
+     by = 10), method = "simple", SNR = 6)
```

The peaks must be aligned using `peakAlign`. Below, the mean spectrum of the raw data is used as the reference, so the peaks will be aligned to the local maxima in the mean spectrum.

```
> pig206.peaklist <- peakAlign(pig206.peaklist, ref = pig206.norm,
+     method = "diff", units = "ppm", diff.max = 200)
```

Below, we use the `peakFilter` method to drop peaks that occur less frequently than once every 100 spectra.

```
> pig206.peaklist <- peakFilter(pig206.peaklist, method = "freq",
+     freq.min = ncol(pig206.peaklist)/100)
```

Finally, `reduceDimension` method is used to sweep back through the full normalized dataset retrieve the identified peaks from all of the pixels.

```
> pig206.peaks <- reduceDimension(pig206.norm, ref = pig206.peaklist,
+     type = "height")

> summary(pig206.peaks)

Class: MSImageSet
Features: m/z = 151.33 ... m/z = 889.67 (143 total)
Pixels: x = 72, y = 1 ... x = 83, y = 66 (4959 total)
x: 10 ... 120
y: 1 ... 66
Size in memory: 6.6 Mb
```

An alternative pre-processing workflow would be to perform peak-picking on all mass spectra and use these peaks directly (after alignment) rather than use `reduceDimension`. However, this would result in 0 intensities for mass spectra where certain peaks were not found, so it places a greater burden on the accuracy of the peak detection algorithm.

Visualization of Molecular Ion Images

Plotting ion images is the natural first step in exploring a mass spectrometry imaging dataset. The ion images shown in Figure 4.1B and Figure 4.1C can plotted using *Cardinal* with the following code.

```
> image(pig206, mz = 888.67, contrast.enhance = "histogram",
+     smooth.image = "gaussian")
```

```
> image(pig206, mz = 186.42, contrast.enhance = "histogram",
+     smooth.image = "gaussian")
```

Segmentation Using Spatial Shrunken Centroids

This section demonstrates the spatial shrunken centroids segmentation method (as described in Chapter 3 and Chapter 4) for statistical analysis implemented in *Cardinal* .

The parameters to be explicitly provided in the `spatialShrunkenCentroids` method are:

- r: The neighborhood smoothing radius
- k: The initial number of segments (clusters)
- s: The shrinkage parameter

For a detailed explanation of the parameters, see Section 3.2.6.

Below, we perform spatial shrunken centroids segmentation with the `method="gaussian"` weights.

```
> set.seed(1)
> pig206.sscg <- spatialShrunkenCentroids(pig206.peaks, r = c(1, 2),
+     k = c(15, 20), s = c(0, 3, 6, 9), method = "gaussian")
```

```
> summary(pig206.sscg)
```

```
   r  k s   method   time Predicted # of Classes
1  1 15 0 gaussian 14.557                     15
2  1 15 3 gaussian 25.602                     10
3  1 15 6 gaussian 32.892                      7
4  1 15 9 gaussian 19.852                      6
5  1 20 0 gaussian 20.906                     19
6  1 20 3 gaussian 28.911                     11
7  1 20 6 gaussian 27.199                      8
8  1 20 9 gaussian 36.020                      6
9  2 15 0 gaussian 55.460                     13
10 2 15 3 gaussian 42.302                     10
11 2 15 6 gaussian 40.441                      6
12 2 15 9 gaussian 42.164                      6
13 2 20 0 gaussian 48.375                     18
14 2 20 3 gaussian 77.894                      9
15 2 20 6 gaussian 50.536                      6
16 2 20 9 gaussian 43.045                      6
   Mean # of Features per Class
1                           143
2                            91
3                            77
4                            63
5                           143
6                            92
7                            74
8                            62
9                           143
10                           90
11                           82
12                           62
13                          143
14                           90
15                           82
16                           60
```

We perform spatial shrunken centroids segmentation with adaptive weights by setting `method="adaptive"` weights.

The resulting object has sixteen sets of model parameters, in the parameter space of $r = 1, 2$, $k = 15, 20$, and $s = 0, 3, 6, 9$.

As seen in the summaries above, many of the segmentations result in fewer numbers of segments than at initialization, and the number of segments is generally lower for higher sparsity. This can be used to determine the number of segments, as described in Section 4.2.2.

Plotting the Spatial Segmentations

We will plot four of the spatial segmentations for the Gaussian weights with different levels of sparsity. This is specified by the `model` argument, where we can list the parameters for the models we would like to plot.

```
> image(pig206.sscg, model = list(r = 2, k = 20, s = c(0, 3, 6, 9)),
+     key = FALSE, layout = c(2, 2))
```

This plots the four segmentations shown in Figure 4.9 from Section 4.2.2.

Plotting and Interpreting the t-statistics of the m/z Values

As described in Section 4.2.3, an important goal of our approach to spatial segmentation is that we not only want a meaningful segmentation, but we also want to be able to identify and rank the important mass features that inform that segmentation. The `spatialShrunkenCentroids` method produces t-statistics for this purpose.

For each mass feature (m/z value), t-statistics are calculated for each segment as described in Section 3.2.2, by comparison to the global mean spectrum.

Positive t-statistics correspond to systematic enrichment in that segment. Negative t-statistics correspond to systematic absence from that segment. The shrinkage parameter s is used to shrink t-statistics toward 0, and when a t-statistic is set to 0, that mass feature is no longer used to determe segment membership.

The t-statistics for the heart segment and the liver segment, as shown in Figure 4.11E and Figure 4.11F from Section 4.2.3, can be plotted as follows.

```
> plot(pig206.sscg, mode = "tstatistics", model = list(r=2, k=20, s=6),
+     key = FALSE, column = 5, ylab = "liver t-statistics")
```

```
> plot(pig206.sscg, mode = "tstatistics", model = list(r=2, k=20, s=6),
+     key = FALSE, column = 6, ylab = "heart t-statistics")
```

In this case, the 5th segment corresponds to the heart, and the 6th segment corresponds to the liver.

Plotting the t-statistics reveals that the liver segment has many more mass features associated with it compared to the heart segment.

The top m/z values for a segmentation can be queried using the `topLabels` method.

```
> topLabels(pig206.sscg, n = 10)

         mz r  k s classes   centers tstatistics
1  269.3333 2 20 0      11 177.93947    49.26370
2  269.3333 2 15 0      11 178.60039    48.95050
3  269.3333 1 20 0      11 167.71484    48.59475
4  269.3333 1 15 0      11 168.12032    48.50941
5  537.2500 2 15 0       5  31.48324    47.73488
6  537.2500 1 15 0       5  31.72642    47.64653
7  269.3333 1 15 3       4 165.49849    47.43609
8  269.3333 1 20 3      11 165.49371    47.41058
9  563.2500 2 15 0       5  28.73309    47.40687
10 563.2500 1 15 0       5  28.93201    47.25001
   p.values adj.p.values
1         0            0
2         0            0
3         0            0
4         0            0
5         0            0
6         0            0
7         0            0
8         0            0
9         0            0
10        0            0
```

This list can be filtered by the segment, model parameters, etc.

```
> topLabels(pig206.sscg, model = list(r = 2, s = 6, k = 20),
+     filter = list(classes = 5))

        mz r  k s classes  centers tstatistics p.values adj.p.values
1 537.2500 2 20 6       5 27.35050    42.19177        0            0
2 563.2500 2 20 6       5 24.70218    41.70469        0            0
3 535.2500 2 20 6       5 19.56177    39.62103        0            0
```

```
4 887.6667 2 20 6       5 38.07015    31.04249         0             0
5 509.2500 2 20 6       5 12.68780    30.56082         0             0
6 281.6667 2 20 6       5 57.27400    29.44007         0             0
```

This makes it easy to rank the most important mass features for distinguishing each segment.

Identifying the Number of Segments

As described in Section 4.2.2, a unique property of spatial shrunken centroids segmentation is that it facilitates a natural way to identify an appropriate number of segments for a segmentation. We do this by plotting the number of predicted segments against the shrinkage parameter s as shown Figure 4.9A from Section 4.2.2.

```
> plot(summary(pig206.sscg), main = "Number of segments")
```

In Figure 4.9A, we look for the shrinkage parameter s for which the predicted number of segments match up between different initialized numbers of segments k. For $r = 2$, this occurs at $s = 6$.

Therefore, we choose the segmentation with Gaussian weights with $r = 2, k = 20, r = 6$ for further exploration. This segmentation is plotted with custom colors in Figure 4.8E in Section 4.2.1.

```
> mycol <- c(internal1 = "#FD9827", back = "#42FD24",
+     internal2 = "#1995FC", brain = "#FC23D9",
+     liver = "#3524FB", heart = "#FC0D1B", bg = "#CDFD34")
> image(pig206.sscg, model = list(r = 2, k = 20, s = 6), key = FALSE,
+     col = mycol, main = "SA +  Shrunken Centroids")
```

Typically, we recommend choosing the segmentation with the most retained features (least sparsity) after which the predicted number of segmentations become approximately equal between different initializations of k.

5.3.2 Supervised Classification Workflow

Here we show how *Cardinal* was used to analyze the RCC dataset shown in Figure 4.4 and produce the classification results shown in Figure 4.14 from Section 4.1. The RCC dataset consists of 8 matched pairs of human kidney tissue. Each tissue pair consists of a normal tissue sample and a cancerous tissue sample. The goal of the workflow is to develop classifiers for predicting whether a new tissue sample is normal or cancer.

```
> library(CardinalWorkflows)
> data(rcc, rcc_analyses)
```

In this dataset, we expect that normal tissue and cancerous tissue will have unique chemical profiles, which we can use to classify new tissue based on the mass spectra.

```
> summary(rcc)
```

```
Class: MSImageSet
Features: m/z = 150.08 ... m/z = 1000 (10200 total)
Pixels: x =  1, y = 13, sample = MH0204_33 ... x = 77, y =  5, sample
 = UH9912_01 (16000 total)
x: 1 ... 99
y: 1 ... 38
Size in memory: 629.1 Mb
```

As can be seen in Figure 4.4 in Section 4.1, each matched pair of tissues belonging to the same subject are on the same slide. Note also the the cancer tissue is on the left and the normal tissue is on the right on each slide.

The image contains 16000 pixels with 10200 spectral features measured at each location (m/z range from 150 to 1000).

Normalization

Before resampling or binning, normalization is necessary to correct for pixel-to-pixel variation. We will use TIC normalization, which is a popular choice for mass spectrometry imaging datasets, as described in Section 2.1.2.

```
> rcc.norm <- normalize(rcc, method = "tic")
```

Resampling to Unit Resolution

The normalized data is then resampled to unit resolution. Binning would also be an appropriate alternative, and could be used by setting `method="bin"` in the `reduceDimension` method.

```
> rcc.resample <- reduceDimension(rcc.norm, method = "resample")
```

Resampling or binning is preferred to peak-picking for classification, in order to avoid bias in cross-validation. However, if peak-picking is preferred, this can be worked around by performing peak-picking separately on the training set *only*, and using the same peaks in the testing and validation sets. This can become a complex procedure if cross-validation is desired.

Subsetting the Dataset

We will subset the dataset to drop pixels that contain only the slide background, so that the final dataset will only consist of mass spectra from actual tissue.

To subset the data, we will use the `diagnosis` variable stored in the object's `pixelData`. This variable is a *factor* with the disease condition for each pixel, as annotated by a pathologist.

```
> summary(rcc$diagnosis)
```

```
cancer normal   NA's
  2775   3302   9923
```

We drop the 9923 pixels without annotation.

```
> rcc.small <- rcc.resample[,rcc$diagnosis %in% c("cancer", "normal")]
> summary(rcc.small)
```

```
Class: MSImageSet
Features: m/z = 151 ... m/z = 1000 (850 total)
Pixels: x = 17, y = 15, sample = MH0204_33 ... x = 61, y =  6, sample
 = UH9912_01 (6077 total)
x: 2 ... 91
y: 2 ... 37
Size in memory: 41.6 Mb
```

Now the dataset contains only the 6077 mass spectra we need to train and test a classifier.

Visualization of Molecular Ion Images

To begin visualizing the dataset, we will plot ion images for m/z values we already know to be useful in distinguishing normal tissue versus cancer.

The ion images for m/z 215.3, known to be more abundant in normal tissue (right) [2], as shown in Figure 4.5 from Section 4.1, can be plotted as follows.

```
> image(rcc, mz = 215.3, normalize.image = "linear",
+     contrast.enhance = "histogram", smooth.image = "gaussian",
+     layout = c(4, 2))
```

Likewise, the ion images for m/z 885.7, known to be more abundant in cancerous tissue (left) [2], as shown in Figure 4.6 from Section 4.1, can be plotted as follows.

```
> image(rcc, mz = 885.7, normalize.image = "linear",
+     contrast.enhance = "histogram", smooth.image = "gaussian",
+     layout = c(4, 2))
```

From Figure 4.6 and Figure 4.5, we note that there is still a great deal of variation in these images for ions that should be associated with a particular disease condition. For example, m/z 215.3 – which should be more abundant in normal tissue – is also abundant in cancerous tissue for samples UH0505_12 and UH9905_18. This shows that multiple ions will be necessary for classification.

Classification Using Spatial Shrunken Centroids

This section demonstrates the spatial shrunken centroids classification method (as described in Chapter 3 and Chapter 4) as implemented in *Cardinal* .

The parameters to be explicitly provided in the `spatialShrunkenCentroids` method are:

- r: The neighborhood smoothing radius

- s: The shrinkage parameter

For a more detailed explanation of these parameters, see Section 3.2.6.

Cross-validation with Spatial Shrunken Centroids

An important step in classification is testing and validation. Therefore, *Cardinal* implements the `cvApply` method, which performs cross-validation for any of the supplied classification methods, including `PLS`, `OPLS`, and |spatialShrunkenCentroids|.

By default, `cvApply` considers each unique sample (as given by the `sample` variable in an `MSImageSet` object's `pixelData`) as a fold for n-fold cross-validation. In most cases, these should correspond to biological replicates, which is our recommended workflow.

This is the case for the RCC dataset, where each matched pair on a separate slide constitutes a unique sample.

```
> summary(rcc.small$sample)
```

```
MH0204_33 UH0505_12 UH0710_33 UH9610_15 UH9812_03 UH9905_18 UH9911_05
      811       394       363       801       756       614       937
UH9912_01
     1401
```

Generally, biological replicates should be used to partition the dataset rather than technical replicates or individual pixels. The only exception would be in the case of a sample size of one, in which case there are no biological replicates. However, a sample size of one is a worst case scenario, and biological replicates should always be preferred.

Below, we perform cross-validation with spatial shrunken centroids classification and the `method="gaussian"` weights.

```
> rcc.cv.sscg <- cvApply(rcc.small, .y = rcc.small$diagnosis,
+     .fun = "spatialShrunkenCentroids", method = "gaussian",
+     r = c(1, 2, 3), s = c(0, 4, 8, 12, 16, 20, 24, 28))
```

We could also perform cross-validation with spatial shrunken centroids classification with adaptive weights by setting `method="adaptive"` weights.

Below, we plot the cross-validated accuracy for the classifier with Gaussian weights, as shown in Figure 4.15 from Section 4.2.5.

```
> plot(summary(rcc.cv.sscg))
```

As shown in Figure 4.15, for all smoothing radii r, the highest accuracy occurs with a shrinkage parameter $s = 20$. For Gaussian weights with $r = 3, s = 20$, accuracy was 88.8%.

Note that in general, the accuracy increases with larger smoothing neighborhood radii r. This is true in this case because rather than heterogenous samples with both normal and cancerous cells on the same tissue, each tissue is relatively homogenous with predominantly normal or cancerous cells. Therefore, greater spatial smoothing increases the accuracy, and adaptive weights would have no advantage over Gaussian weights. For classification on more heterogenous tissue, adaptive weights and smaller neighborhod radii may perform better.

Plotting the Classified Images

Below, the classified images for Gaussian weights, as shown in Figure 4.14 from Section 4.2.5, are plotted.

```
> image(rcc.cv.sscg, model = list(r = 3, s = 20), layout = c(4, 2))
```

Spatial shrunken centroids produce probabilities of cancer versus normal, which we plot using higher opacity for higher probability. This makes for more interpretable predicted images than non-probabilistic classifiers.

Plotting and Interpreting the t-statistics of the m/z Values

To inspect the t-statistics of the m/z values, we now train a classifier on the full dataset using the parameters $r = 3, s = 20$.

```
> rcc.sscg <- spatialShrunkenCentroids(rcc.small, y =
+     rcc.small$diagnosis, r = 3, s = 20, method = "gaussian")
```

Below, we show how to plot the shrunken centroids and the t-statistics, as shown in Figure 4.16 from Section 4.2.5.

```
> plot(rcc.sscg, mode = "centers", model = list(r = 3, s = 20),
+     column = "cancer")

> plot(rcc.sscg, mode = "centers", model = list(r = 3, s = 20),
+     column = "normal")

> plot(rcc.sscg, mode = "tstatistics", model = list(r = 3, s = 20))
```

As seen in Figure 4.16, only a few m/z values have non-zero t-statistics.

```
> summary(rcc.sscg)

  r k  s   method  time Predicted # of Classes
1 3 2 20 gaussian 3.128                      2
  Mean # of Features per Class
1                           40
```

In fact, only 40 of 850 mass features are used in the spatial shrunken centroids classifier.

We identify the top-ranked mass features using the `topLabels` method.

```
> topLabels(rcc.sscg)

   mz r k  s classes   centers tstatistics p.values adj.p.values
1 215 3 2 20  normal  5.955134    18.83852        0            0
2 886 3 2 20  cancer 19.711863    15.90639        0            0
3 810 3 2 20  normal  8.640044    13.79732        0            0
4 751 3 2 20  cancer  4.030214    12.62721        0            0
5 279 3 2 20  normal  3.596872    12.54607        0            0
6 353 3 2 20  normal  2.261867    11.95248        0            0
```

Spatial shrunken centroids identified m/z 215, m/z 886, and m/z 810, which are all known to be important in disinguishing cancer from normal in RCC [2].

6. MATTER: OPEN-SOURCE SOFTWARE FOR LARGE COMPLEX DATASETS ON DISK

6.1 Overview of *matter*

matter is a free and open-source R package for the rapid development of statistical methods for large experimental datasets on disk. It is aimed at statistical analysis of datasets that do not fit into computer memory, and which may be stored in domain-specific binary formats, such as high-resolution MS imaging experiments, which are commonly stored as Analyze 7.5 or imzML [9]. It differs from similar R packages aimed at analysis of large datasets in that it is focused on flexibility and applicability to a wide variety of experimental file formats. It is designed for rapid adaptation of statistical methods to data stored in new file formats.

matter contributes scalable, flexible statistical computing infrastructure for the analysis of larger-than-memory experimental datasets in custom file formats, including support for statistical analysis of high-resolution, high-throughput MS imaging experiments.

6.1.1 Necessity of Scalability

Scalability is a growing concern for statistical analysis of MS imaging experiments. Consistent improvements in instrumentation have led to rapid increases in mass and spatial resolutions, leading to dramatically larger datasets. Due to differing experimental requirements, the largest MS imaging datasets can often be several orders of magnitude larger than smaller experiments. Where smaller experiments may result in datasets on the order of 100 MB per sample, higher-resolution experiments can produce datasets on the order of 100 GB per sample.

In addition to increasing mass and spatial resolutions, experimental complexity is also increasing. More experiments are now utilizing multiple samples, and as sample preparation is continually improved and perfected, these sample sizes are increasing, too. With the wider prevalence of 3D imaging, it is also more common for each sample in the experiment to span multiple 2D images.

Increasing mass resolution, spatial resolution, and sample sizes mean that many experiments now span multiple files, and those files are becoming larger. It is necessary for statistical software and methods to scale to these new requirements.

This is a problem for many existing statistical software packages in R which assume that the dataset fits entirely into memory. Although some languages offer advantages in this area, they are less efficient than R in terms of development time for statistical methods, because they do not offer comparable resources for statistical programming. However, the existing R packages that are designed to work with large datasets, such as `bigmemory` and `ff`, often require converting the dataset to a new file format, or have strict requirements on the file format [28, 29]. In domains such as MS imaging, this can be a major burden on statisticians and experimentalists who must already convert many large data files from a proprietary vendor-specific format to an open format such as imzML. It is especially burdensome considering that the data will likely still need to be processed (e.g., with normalization, baseline reduction, peak picking, etc.), before any statistical analysis can take place.

This problem is not limited to MS imaging, but applies to statistics in general. Scalability must be a major concern for development of new statistical methods if statistics is to remain relevant in the era of big data. Existing packages like `bigmemory` and `ff` have greatly simplified the problem for statisticians, but still require additional care when using them due to the limitations they impose to achieve fast computation. However, the success of R has shown that the flexibility to facilitate rapid prototyping of new statistical methods is often more important than premature optimization. This is especially important when developing statistical methods for experimental applications dominated by large datasets and unique file formats. It is important to be able to demonstrate statistical results on the datasets that are most relevant to experimentalists in the area, which may sometimes be too large to load into memory. It is not possible to develop statistical methods for experiments without being able to work directly with the data. For the growing number of fields with big data, working with the data is becoming a challenge in itself.

matter proposes to solve these problems by providing a flexible infrastructure for statistical computing with datasets on disk. It is customizable to different binary file formats, and allows direct access to the on-disk data for either processing or statistical analysis, so that additional file conversion is not necessary. *matter* enables visualization and statistical analysis of complex, larger-than-memory datasets stored in an arbitrary number of files of any size. In particular, it allows *Cardinal* to analyze high-resolution, high-throughput MS imaging experiments.

6.1.2 Applicability and Requirements

matter is applicable to datasets stored on disk, in any number of files, in any open-source binary file format, including the imzML and Analyze 7.5 formats for MS imaging experiments. *matter* has been tested on imzML datasets up to 26.4 GB in size. *matter* is compatible with Windows, Mac, and Linux operating systems. There are no specific memory requirements on the total size of data, but available memory should be twice the size of the largest segment of data required to be accessed at once for a single calculation. Most calculations can operate on small segments of the data at a time. *matter* runs optimally with contiguous data stored on a fast storage device, such as a solid state drive (SSD), but these are not requirements.

6.1.3 Functionalities for Statistical Analysis

For summary statistics, *matter* provides methods for memory-efficient calculation of mean and variance for `matter` objects. Variance is calculated using the method of Welford [30], which has been shown to be accurate for large floating-point datasets, while requiring only a single data pass instead of two. For other statistical calculations, *matter* also provides its own `apply` method for user-specified operations on rows and columns of its on-disk matrices, while loading only a single row or column into memory at a time.

For statistical modeling, *matter* provides an interface to the *biglm* package, which implements memory-efficient linear regression and fitting of generalized linear models [31]. The `bigglm` function from the *biglm* package requires as input a function which retrieves the next chunk of the data, and *matter* provides a wrapper function so that `matter` matrices can be treated as an ordinary `data.frame` for linear model fitting.

matter opens the possibility for many statistical approaches to be applied to larger-than-memory data. For example, because *matter* implements basic linear algebra for on-disk matrices, iterative methods that operate only on small portions of the data at once, such as the implicitly restarted Lanczos bidiagonalization algorithm (IRLBA) for eigendecomposition of large matrices, can be applied directly to on-disk `matter` matrices with the `irlba` package to perform principal components analysis (PCA) on larger-than-memory datasets [32, 33]. *matter* enables these approaches to be applied with other statistical methods such as PLS-DA and O-PLS-DA.

6.1.4 Implementation and Performance

matter is designed around the idea of contiguous ("atomic") sectors of disk, by analogy to R's notion of "atomic" vectors, which are the building blocks of R's more complex data structures like lists, `data.frame`s, etc. Likewise, *matter* is organized around the idea of "atomic" sectors of disk that are part of a larger dataset. A `matter` object is a vector or matrix defined by user-specified locations of the dataset on disk. In the simplest possible case, this could be a vector stored contiguously in a single file. In a more complex case, it could be a matrix where each row or column is stored in a separate file. However, neither vectors nor the rows and columns of matrices need to be stored contiguously, or even in the same file.

matter uses memory-efficient data structures so that only the portion of the dataset that is necessary for a calculation is loaded into memory, and then freed after that calculation completes. This allows for a minimal memory footprint, at the cost of heavy disk use. *matter* compensates for this by attempting to utilize sequential read/writes over random read/writes whenever possible, while minimizing the total number of atomic read/write operations. For example, by using *matter* matrices with the *biglm* package, fitting a generalized linear model to a 1.2 GB dataset used only 468 MB of total memory and took 49 seconds on a 2012 MacBook Pro 2.6 GHz with SSD.

matter is intended for rapid prototyping of statistical methods for larger-than-memory on-disk datasets, and therefore focuses on minimizing the amount of developer effort that must be devoted to thinking about dataset management and computational concerns. In particular, it is intended for domain-specific applications where existing binary file formats are already in use for storage of large experimental datasets. When computational performance becomes a priority, related packages such as *bigmemory* or *ff*, which place more stringest requirements on data structure to allow for greater computational optimizations, may be preferable [28, 29].

6.2 Design and Implementation of *matter*

matter is designed with several goals in mind. Like the *bigmemory* and *ff* packages, it seeks to make statistical methods scalable to larger-than-memory datasets by utilizing data-on-disk. Unlike those packages, it seeks to make domain-specific file formats (such as Analyze 7.5 and imzML for MS imaging experiments) accessible from disk directly without additional file conversion. It seeks to have a minimal memory footprint, and require minimal developer effort to use, while maintaining computational efficiency wherever possible.

6.2.1 S4 Classes

matter utilizes S4 classes to implement on-disk matrices in a way so that they can be seamlessly accessed as if they were ordinary R matrices. These are the `atoms` class and the `matter` class. The `atoms` class is not exported to the user, who only interacts with the `matter` class and `matter` objects to create and manipulate on-disk matrices.

`atoms`: Contiguous Sectors of Data on Disk

By analogy to R's notion of "atomic" vectors, the `atoms` class uses the notion of contiguous "atomic" sectors of disk. Each "atom" in an `atoms` object gives the location of one block of contiguous data on disk, as denoted by a file path, an byte offset from the beginning of the file, a data type, and the number of data elements (i.e., the length) of the atom. An `atoms` object may consist of many atoms from multiple files and multiple locations on disk, while ultimately representing a single vector or row or column of a matrix.

Structure:

- `length`: the number of atoms in the object
- `file_id`: the ID's of the files where each atom is located
- `datamode`: the type of data (short, int, long, float, double) for each atom
- `offset`: each atom's byte offset from the beginning of the file
- `extent`: the length of each atom
- `index_offset`: the cumulative index of the first element of each atom
- `index_extent`: the cumulative one-past-the-end index of each atom

The `atoms` class has a C++ backend in the `Atoms` C++ class.

`matter`: Vectors and Matrices Stored on Disk

A `matter` object is made of one or more `atoms` objects, and represents a vector or matrix. It includes additional metadata such as dimensions and row names or column names.

Structure:

- `data`: one or more `atoms` objects
- `datamode`: the type of data (integer, numeric) for the represented vector or matrix
- `filepath`: the paths to the files used by the `atoms` objects
- `filemode`: should the files be open for read/write, or read-only?
- `chunksize`: how large the chunk sizes should be for calculations that operate on chunks of the dataset
- `length`: the total length of the dataset
- `dim`: the extent of each dimension (for a matrix)
- `names`: the names of the data elements (for a vector)
- `dimnames`: the names of the dimensions (for a matrix)

A `matter_vec` vector contains a single `atoms` object that represents all of the atoms of the vector. The `matter_mat` matrix class has two subtypes for column-major (`matter_matc`) and row-major (`matter_matr`) matrices. A column-major `matter_matc` matrix has one `atoms` object for each column, while a row-major `matter_matr` matrix has one `atoms` object for each row.

The `matter` class has a C++ backend in the `Matter` C++ class.

6.2.2 C++ Classes

matter utilizes a C++ backend to access the data on disk and transform it into the appropriate representation in R. Although these classes correspond to S4 classes in R, and are responsible for most of the computational work, all of the required metadata is stored in the S4 classes in R, and are simply read by the C++ classes. This means that *matter* never depends on external pointers, which makes it trivial to share `matter` vectors and `matter` matrices between R sessions that have access to the same filesystem.

`Atoms`: Contiguous Sectors of Data on Disk

The `Atoms` C++ class is responsible for reading and writing the data on disk, based on its metadata. For computational efficiency, it tries to perform sequential read/writes rather than random read/writes whenever possible, while minimizing the total number of atomic read/writes on disk.

`Matter`: Vectors and Matrices Stored on Disk

The `Matter` C++ class is responsible for transforming the data read by the `Atoms` class into a format appropriate for R. This may include re-arranging contiguous data that has been read sequentially into a different order, either due to the inherent organization of the dataset, or as requested by the user in R.

`MatterAccessor`: Iterate over Virtual Disk Objects

The `MatterAccessor` C++ class acts similarly to an iterator, and allows buffered iteration over a `Matter` object. It can either iterate over the whole dataset (for both vectors and matrices), or over a single column for column-major matrices, or over a single row for row-major matrices.

A `MatterAccessor` object will load portions of the dataset (as many elements as the `chunksize` at once) into memory, and then free that portion of the data and load a new chunk, as necessary. This buffering is handled automatically by the class, and code can treat it as a regular iterator. This allows seamless and simple iteration over `Matter` objects while maintaining strict control over the memory footprint of the calculation.

6.3 *matter* Examples

6.3.1 Example 1: Attaching and Working with On-disk Matrices

matter matrices and vectors can be initialized similarly to ordinary R matrices. When no file is given, a new temporary file is created in the default temporary file directory, which will be cleaned up later by either R or the operating system.

Here, we initialize a *matter* matrix with 10 rows and 10 columns. The resulting object is a subclass of the `matter` class, and stores file metadata that gives the location of the data on disk. In many cases, it can be treated as an ordinary R matrix.

```
> x <- matter_mat(data=1:50, nrow=10, ncol=5)
> x

An object of class 'matter_matc'
  <10 row, 5 column> on-disk binary matrix
    files: 1
    datamode: numeric
    16.5 KB in-memory
    400 bytes on-disk

> x[]

      [,1] [,2] [,3] [,4] [,5]
 [1,]    1   11   21   31   41
 [2,]    2   12   22   32   42
 [3,]    3   13   23   33   43
 [4,]    4   14   24   34   44
 [5,]    5   15   25   35   45
 [6,]    6   16   26   36   46
 [7,]    7   17   27   37   47
 [8,]    8   18   28   38   48
 [9,]    9   19   29   39   49
[10,]   10   20   30   40   50
```

As seen above, this is a small toy example in which the in-memory metadata actually takes up more space than the size of the data stored on disk. For much larger datasets, the in-memory metadata will be a small fraction of the total size of the dataset on disk.

matter 's matrices and vectors can be indexed into like ordinary R matrices and vectors.

```
> x[1:4,]

     [,1] [,2] [,3] [,4] [,5]
[1,]    1   11   21   31   41
[2,]    2   12   22   32   42
[3,]    3   13   23   33   43
[4,]    4   14   24   34   44

> x[,3:4]

      [,1] [,2]
 [1,]   21   31
 [2,]   22   32
 [3,]   23   33
 [4,]   24   34
 [5,]   25   35
 [6,]   26   36
 [7,]   27   37
 [8,]   28   38
 [9,]   29   39
[10,]   30   40
```

We can assign names to `matter_vec` vectors and row and column names to `matter_mat` matrices.

```
> rownames(x) <- 1:10
> colnames(x) <- letters[1:5]
> x[]
```

```
    a  b  c  d  e
1    1 11 21 31 41
2    2 12 22 32 42
3    3 13 23 33 43
4    4 14 24 34 44
5    5 15 25 35 45
6    6 16 26 36 46
7    7 17 27 37 47
8    8 18 28 38 48
9    9 19 29 39 49
10  10 20 30 40 50
```

matter provides methods for calculating summary statistics for its vectors and matrices, including some methods that do not exist in base R, such as `colVar`.

```
> colSums(x)

  a   b   c   d   e
 55 155 255 355 455

> colSums(x[])

  a   b   c   d   e
 55 155 255 355 455

> colVar(x)

       a        b        c        d        e
9.166667 9.166667 9.166667 9.166667 9.166667

> apply(x, 2, var)

       a        b        c        d        e
9.166667 9.166667 9.166667 9.166667 9.166667
```

One of the major advantages of the flexibility of *matter* is being able to treat data from multiple files as a single dataset. This is particularly useful if analysing data from a domain where each sample in an experiment generates large files, such as high-resolution, high-throughput mass spectrometry imaging.

Below, we create a second matrix, and show its data is stored in a separate file. We then combine the matrices, and the result can be treated as a single matrix, despite originating from multiple files. Combinging the matrices does not create new data or change the existing data on disk.

```
> y <- matter_mat(data=51:100, nrow=10, ncol=5)
> filepath(x)
```

```
[1] "/var/folders/bf/tcngbyjj6kn540c74_kf6ypw0000gp/T//Rtmpqjdp9A/
fileede37a87e46d.bin"
```

```
> filepath(y)
```

```
[1] "/var/folders/bf/tcngbyjj6kn540c74_kf6ypw0000gp/T//Rtmpqjdp9A/
fileede313cb3744.bin"
```

```
> z <- cbind(x, y)
> z
```

```
An object of class 'matter_matc'
  <10 row, 10 column> on-disk binary matrix
    files: 2
    datamode: numeric
    33 KB in-memory
    800 bytes on-disk
```

```
> z[]
```

```
    a  b  c  d  e
1   1 11 21 31 41 51 61 71 81  91
2   2 12 22 32 42 52 62 72 82  92
3   3 13 23 33 43 53 63 73 83  93
4   4 14 24 34 44 54 64 74 84  94
5   5 15 25 35 45 55 65 75 85  95
6   6 16 26 36 46 56 66 76 86  96
7   7 17 27 37 47 57 67 77 87  97
8   8 18 28 38 48 58 68 78 88  98
9   9 19 29 39 49 59 69 79 89  99
10 10 20 30 40 50 60 70 80 90 100
```

Note that matrices in *matter* are either stored in a column-major or a row-major format. The default is to use the column-major format, as R does. Column-major matrices are optimized for fast column-access, and assume that each column is stored contiguously or mostly-contiguously on disk. Conversely, row-major matrices are optimized for fast row-access, and make the same assumption for rows.

Since *matter* does support both column-major and row-major formats, transposing a matrix is a trivial operation in *matter* that only needs to change the matrix metadata, and doesn't touch the data on disk.

```
> t(x)

An object of class 'matter_matr'
  <5 row, 10 column> on-disk binary matrix
    files: 1
    datamode: numeric
    16.8 KB in-memory
    400 bytes on-disk

> rbind(t(x), t(y))

An object of class 'matter_matr'
  <10 row, 10 column> on-disk binary matrix
    files: 2
    datamode: numeric
    33 KB in-memory
    800 bytes on-disk
```

Note that this is equivalent to `t(cbind(x, y))`.

Below, we inspect the metadata associated with the different columns of `x`.

```
> x@data

[[1]]
  file_id datamode offset extent index_offset index_extent
1       1   double      0     10            0           10

[[2]]
  file_id datamode offset extent index_offset index_extent
1       1   double     80     10            0           10

[[3]]
  file_id datamode offset extent index_offset index_extent
1       1   double    160     10            0           10

[[4]]
  file_id datamode offset extent index_offset index_extent
1       1   double    240     10            0           10

[[5]]
  file_id datamode offset extent index_offset index_extent
1       1   double    320     10            0           10
```

Note that each column has a byte offset and an extent (i.e., length) associated with it.

Now we show how to create a `matter_mat` matrix for an pre-existing file. We will point the new matrix to the bottom half of `x`.

```
> xsub <- matter_mat(offset=c(40, 120, 200, 280, 360),
+             extent=rep(5,5), filepath=filepath(x))
> x[6:10,]
```

```
    a  b  c  d  e
6   6 16 26 36 46
7   7 17 27 37 47
8   8 18 28 38 48
9   9 19 29 39 49
10 10 20 30 40 50
```

```
> xsub[]
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    6   16   26   36   46
[2,]    7   17   27   37   47
[3,]    8   18   28   38   48
[4,]    9   19   29   39   49
[5,]   10   20   30   40   50
```

It is possible to build `matter` objects from nearly any possible combination of files and locations within files. It is even possible to build up a matrix from vectors, which we do below.

```
> x2 <- matter_vec(offset=80, extent=10, filepath=filepath(x))
> y3 <- matter_vec(offset=160, extent=10, filepath=filepath(y))
> cbind(x2, y3)[]
```

```
      [,1] [,2]
 [1,]   11   71
 [2,]   12   72
 [3,]   13   73
 [4,]   14   74
 [5,]   15   75
 [6,]   16   76
 [7,]   17   77
 [8,]   18   78
 [9,]   19   79
[10,]   20   80
```

```
> cbind(x[,2], y[,3])

   [,1] [,2]
1    11   71
2    12   72
3    13   73
4    14   74
5    15   75
6    16   76
7    17   77
8    18   78
9    19   79
10   20   80
```

This is a quick and easy way to build a dataset from many files where each column of the dataset is stored in a separate file. Even if the resulting matrix would fit into memory, using *matter* can be a tidy, efficient way of reading complex binary data from multiple files into R.

6.3.2 Example 2: Linear Regression for On-disk Datasets

matter is designed to provide a statistical computing environment for larger-than-memory datasets on disk. To facilitate this, *matter* provides a method for fitting of linear models for `matter` matrices through the *biglm* package. *matter* provides a wrapper for *biglm*'s `bigglm` function that works with `matter_mat` matrices, which we demonstrate below.

First, we simulate some data appropriate for linear regression.

```
> set.seed(81216)
> n <- 15e6
> p <- 9
> b <- runif(p)
> names(b) <- paste0("x", 1:p)
> data <- matter_mat(nrow=n, ncol=p + 1)
> colnames(data) <- c(names(b), "y")
> data[,p + 1] <- rnorm(n)
> for ( i in 1:p ) {
+   xi <- rnorm(n)
+   data[,i] <- xi
+   data[,p + 1] <- data[,p + 1] + xi * b[i]
+ }
> data

An object of class 'matter_matc'
  <15000000 row, 10 column> on-disk binary matrix
    files: 1
    datamode: numeric
    31.2 KB in-memory
    1.2 GB on-disk

> head(data)

              x1        x2         x3          x4         x5
[1,] -0.45330471 0.5995144 -0.1392395  0.36748584  1.4000923
[2,] -1.60355974 0.5862366 -0.5421275 -0.36101120 -0.4930582
[3,]  0.22920974 0.5138377 -1.7860077  1.53126322  0.3557548
[4,] -1.38862865 0.1411892  0.3166607 -0.08396404  0.9629351
[5,] -0.36473656 0.4315282  1.1860328 -1.13518455  0.5386445
[6,] -0.07204838 0.2744724 -0.6730541  0.03472469  0.2138691
      0.5555708         x7          x8         x9          y
[1,]  0.5555708 -2.4031764 -0.57037899 -0.4356390  0.2280728
[2,]  0.7549443 -0.1348020  0.05384544 -0.5209713  0.3358334
[3,] -0.6093811  1.0381120  0.72976777  0.9689488  3.8910764
[4,]  0.3443397 -1.5310565 -0.44875206 -1.1320185 -0.8646491
[5,]  1.1426125  0.2239818  1.40000992 -0.9843404  1.8709778
[6,]  0.5923886  0.4852140 -0.29082018  1.0831832  1.5140973
```

This creates a 1.2 GB dataset on disk, but barely 32 KB of metadata is stored in memory.

Now we calculate some statistical summaries using *matter* 's `apply` method for `matter_mat` matrices.

```
> apply(data, 2, mean)

           x1            x2            x3            x4            x5
 2.962621e-04 -2.596339e-04 -2.729651e-04  3.014581e-05 -5.893552e-05
           x6            x7            x8            x9             y
-2.835383e-04 -1.309537e-04 -9.810476e-05 -1.404680e-04 -3.225581e-04

> apply(data, 2, var)
```

```
       x1        x2        x3        x4        x5        x6        x7
1.0003094 0.9996336 0.9990518 1.0003654 0.9999593 0.9995961 0.9999286
       x8        x9         y
1.0001395 0.9996875 4.4527319
```

We could also have used `colMeans` and `colVar`.

Now we fit the linear model to the data using the `bigglm` method for `matter_mat` matrices. Note that it requires a formula, and (unfortunately) it does not allow `y ~ .`, so all variables must be stated explicitly.

```
> fm <- as.formula(paste0("y ~ ", paste(names(b), collapse=" + ")))
> bigglm.out <- bigglm(fm, data=data, chunksize=floor(n / 2000))
> summary(bigglm.out)
```

```
Large data regression model: bigglm(formula, getNextDataChunk, ...)
Sample size =  1.5e+07
              Coef    (95%    CI)    SE      p
(Intercept) 0.0004 -0.0001 0.0009 3e-04 0.1001
x1          0.1689  0.1684 0.1695 3e-04 0.0000
x2          0.9572  0.9566 0.9577 3e-04 0.0000
x3          0.3801  0.3796 0.3806 3e-04 0.0000
x4          0.6042  0.6037 0.6048 3e-04 0.0000
x5          0.5198  0.5193 0.5203 3e-04 0.0000
x6          0.6926  0.6921 0.6931 3e-04 0.0000
x7          0.8374  0.8369 0.8380 3e-04 0.0000
x8          0.4616  0.4610 0.4621 3e-04 0.0000
x9          0.5782  0.5777 0.5788 3e-04 0.0000
```

```
> cbind(coef(bigglm.out)[-1], b)
```

```
                    b
x1 0.1689408 0.1689486
x2 0.9571547 0.9574388
x3 0.3800765 0.3802078
x4 0.6042379 0.6043915
x5 0.5198087 0.5194832
x6 0.6926179 0.6927430
x7 0.8374374 0.8373628
x8 0.4615518 0.4617963
x9 0.5782414 0.5775168
```

On a 2012 retina MacBook Pro with 2.6 GHz Intel CPU, 16 GB RAM, and 500 GB SSD, fitting the linear model takes 49 seconds and uses an additional 322 MB of memory overhead. The max amount of memory used while fitting the model was only 468 MB, for the 1.2 GB dataset. This shows that fitting the linear model used less memory than the size of the dataset on disk.

While this example used a dataset that could have fit into memory, its shows the memory savings that are possible. Linear regression could still be performed if the dataset were larger than the 16 GB memory of this computer.

6.3.3 Example 3: Principal Components Analysis for On-disk Datasets

Because *matter* provides basic linear algebra for on-disk `matter_mat` matrices with in-memory R matrices, it opens up the possibility for the use of many iterative statistical methods which can operate on only small portions of the data at a time.

For example, `matter_mat` matrices are compatible with the *irlba* package, which performs efficient, bounded-memory singular value decomposition (SVD) of matrices, and which can therefore be used for efficient principal components analysis (PCA) of large datasets [33].

First, we simulate some data appropriate for principal components analysis.

```
> set.seed(81216)
> n <- 15e5
> p <- 100
> data <- matter_mat(nrow=n, ncol=p)
> for ( i in 1:10 )
+   data[,i] <- (1:n)/n + rnorm(n)
> for ( i in 11:20 )
+   data[,i] <- (n:1)/n + rnorm(n)
> for ( i in 21:p )
+   data[,i] <- rnorm(n)
> data
```

```
An object of class 'matter_matc'
  <1500000 row, 100 column> on-disk binary matrix
    files: 1
    datamode: numeric
    281 KB in-memory
    1.2 GB on-disk
```

This again creates a 1.2 GB dataset on disk, but barely 32 KB of metadata is stored in memory. Note that only the first twenty variables show systematic variation, with the first ten varying distinctly from the next ten variables.

First we calculate the variance for each column.

```
> var.out <- colVar(data)
> plot(var.out, type='h', ylab="Variance")
```

This takes only 7 seconds and uses less than 30 KB of additional memory. The maximum amount of memory used while calculating the variance for all columns of the 1.2 GB dataset is only 27 MB.

Now we load the *irlba* package and use it to calculate the first two right singular vectors, which correspond to the first two principal components.

Note that the `irlba` function has an optional argument `mult` which allows specification of a custom matrix multiplication method, for use with packages such as *bigmemory* and *ff*. This is especially useful since it allows a `transpose=TRUE` argument, so that the identity `t(t(B) %*% A)` can be used in place of `t(A) %*% B)` when transposition is an expensive operation. However, this is not necessary for *matter*, since transposition is a trivial operation for `matter_mat` matrices.

```
> library(irlba)
> irlba.out <- irlba(data, nu=0, nv=2)
```

On a 2012 retina MacBook Pro with 2.6 GHz Intel CPU, 16 GB RAM, and 500 GB SSD, calculating the first two principal components takes roughly 2-3 minutes and uses an additional 337 MB of memory overhead. The max amount of memory used during the computation was only 433 MB, for the 1.2 GB dataset. This shows that PCA can be performed using less memory than the size of the dataset on disk, which is necessary if the dataset is larger than memory.

Now we plot the first two principal components.

```
> plot(irlba.out$v[,1], type='h', ylab="PC 1")
```

```
> plot(irlba.out$v[,2], type='h', ylab="PC 2")
```

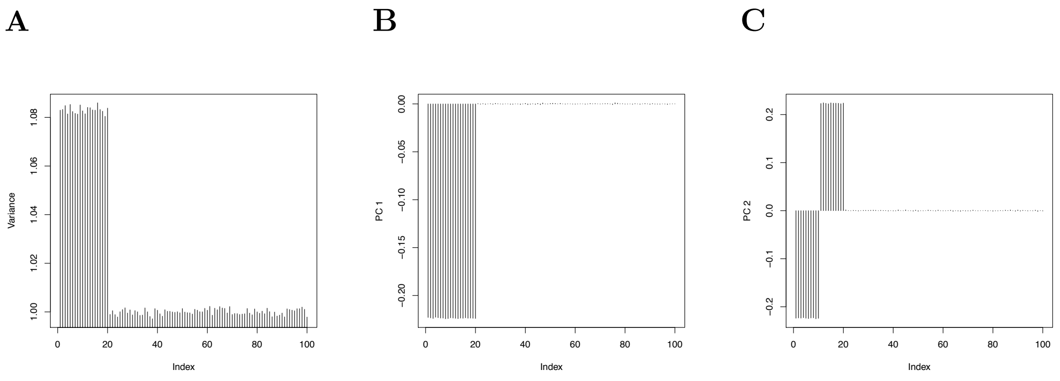


Fig. 6.1. **Principal components analysis of on-disk dataset.** *A*, Sample variance. *B*, PC1 loadings. *C*, PC2 loadings.

As shown in the PCA plots in Figure 6.1, the first PC reveals that most of the variation in the data occurs in the first twenty variables, while the second PC distinguishes the first ten variables from the next ten variables.

As in the previous example, this example again used a dataset that could have fit into memory, but its shows that PCA could still be performed if the dataset were larger than the 16 GB memory of this computer.

6.3.4 Example 4: 3D Mouse Pancreas MS Imaging Dataset

This section demonstrates the usefulness of *matter* for working with large MS imaging experiments in *Cardinal* . For versions >=1.5, *Cardinal* supports using *matter* matrices to access larger-than-memory MS imaging datasets.

We will use one of the benchmark 3D MS imaging experiments from Oetjen *et al.* [34]. We will use the 3D mouse pancreas dataset, which is comprised of 29 tissue sections, with a total of 497,227 pixels and 13,297 features. The data is stored in imzML format [9]. The ".imzML" XML file with experimetnal metadata is 857.7 MB, and the ".ibd" binary file with the m/z values and spectral intensities is 26.45 GB.

Due to the various offsets in imzML ibd files, they cannot be attached as simply as *bigmemory* or *ff* files. These packages have strict requirements on the format of their data, for maximum computational effiency. *matter* takes a different approach with more flexibility, which allows use of imzML's domain-specific binary file format directly, and with minimal memory footprint, at the cost potentially slower computational performance in some situations.

```
> library(matter)
> library(Cardinal)
> path <- "~/Documents/Datasets/MALDI-Imaging/3D_Mouse_Pancreas/"
> file <- "3D_Mouse_Pancreas.imzML"
```

We load the dataset with `readMSIData` with `attach.only=TRUE`. In older versions of *Cardinal* (<1.5), this would use a `Binmat` matrix, which is far less efficient than a `matter` matrix. For newer versions of *Cardinal* (>=1.5), if *matter* is in the search path, then *Cardinal* will use a `matter` matrix.

```
> mouse <- readMSIData(paste0(path, file), attach.only=TRUE)
> summary(mouse)

Class: MSImageSet
Features: m/z = 1591.3 ... m/z = 14317.36 (13297 total)
Pixels: x = 118, y = 72, z = 1 ... x = 138, y = 140, z = 29 (497225
 total)
x: 1 ... 224
y: 1 ... 164
z: 1 ... 29
Size in memory: 1453.1 Mb
```

On a 2012 retina MacBook Pro with 2.6 GHz Intel CPU, 16 GB RAM, and 500 GB SSD, parsing the imzML file and attaching the dataset takes approximately 5 minutes and uses roughly 3.6 GB of memory. This is entirely from parsing the 857.7 MB imzML file. *Cardinal* relies on an XML library to parse the imzML file which requires building a full representation of the XML file in memory, in addition to the overhead of reading the file.

```
> iData(mouse)
```

```
An object of class 'matter_matc'
  <13297 row, 497225 column> on-disk binary matrix
    files: 1
    datamode: numeric
    1.4 GB in-memory
    26.4 GB on-disk
```

As shown above, the matrix metadata takes up approximately 1.4 GB in memory, and points to 26.4 GB on disk. This dataset is larger than the 16 GB of memory of this computer, and could not be loaded at all without using *matter* .

Some *Cardinal* methods can be used normally, such as `pixelApply` and `featureApply`. Note that it is advisable to *avoid* using `featureApply` for large on-disk MS imaging datasets, because the structure of imzML and Analyze 7.5 files make this inefficient. Both file formats stores spectra contiguously, rather than images, so loading images requires many non-contiguous reads, which take much longer to read than contiguous mass spectra.

Nonetheless, we can, for example, use `pixelApply` to calculate the total ion current (TIC) for each pixel.

```
> mouse.tic <- pixelApply(mouse, sum)
> summary(mouse.tic)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  267.4  1104.0  1640.0  2392.0  2736.0 56690.0
```

On the same MacBook Pro, computing the TIC for every pixel takes approximately 4 minutes and uses about 500 MB of additional memory.

3D molecular ion images can be plotted using the `image3D` method introduced in *Cardinal* v1.3.2. We will plot the ion image for m/z 5806, which corresponds to insulin.

```
> image3D(mouse, mz=5806, plusminus=1, phi=45, theta=180)
```

Loading the ion image from file and plotting it takes approximately 2 minutes on the same MacBook Pro and uses an additional 2 GB of memory in overhead. The max amount of memory used while plotting the image was just under 3 GB, for the 26.4 GB dataset. The ion image could not be plotted at all without *matter* , because the dataset is too large to be loaded into memory.

Lastly, we will plot the TIC of each pixel, which we calculated using `pixelApply` above.

```
> image3D(mouse, mouse.tic ~ x * y * z, phi=45, theta=180)
```

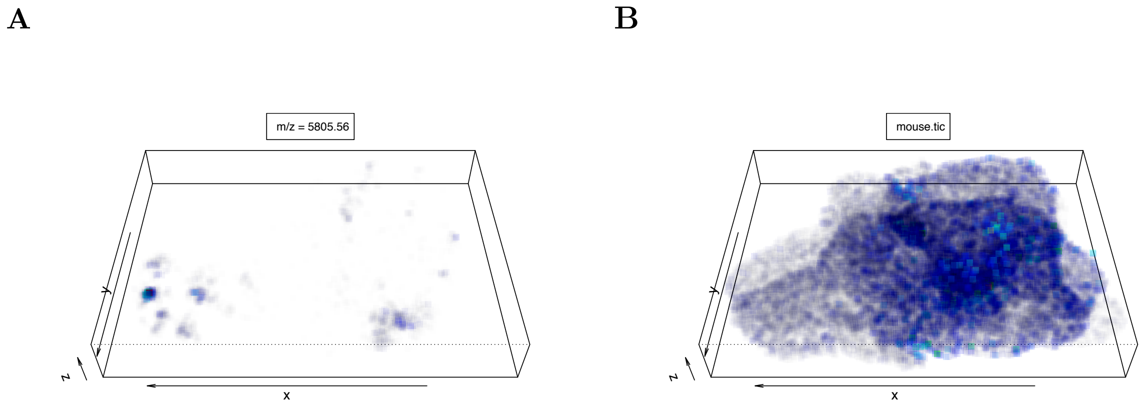


Fig. 6.2. **Benchmark 3D mouse pancreas images.** *A*, m/z 5806 (insulin). *B*, total ion current (TIC).

Using *matter* , it is possible to visualize and analyze datasets that could not be visualized or analyzed before, because they are too large to fit into memory. *matter* solves this by working with the data on disk, allowing us to analyze high-resolution, high-throughput MS imaging experiments that we could not before.

7. SUMMARY AND FUTURE RESEARCH

7.1 Conclusions about Spatial Shrunken Centroids

We proposed spatial shrunken centroids, a regularized statistical framework for segmentation and classification of MS imaging experiments. Spatial shrunken centroids combines the feature selection properties of nearest shrunken centroids with the spatially-aware properties of spatially-aware clustering. The result is a powerful method that delivers results which are comparable to spatially-aware clustering for segmentation, and PLS-DA and O-PLS-DA for classification, but which can also be used for statistical inference.

Its use of statistical regularization aids interpretability by selecting important features, while also creating a clear relationship between sparsity and the number of segments in segmentation, which helps guide the selection of the number of segments. By using opacity to reflect probability when plotting the segmented or classified image, it is also possible to visually characterize the certainty of the segmentation or classification.

However, spatial shrunken centroids does have drawbacks. Although statistical regularization helps to guide the selection of the number of segments, the procedure requires fitting many segmentations, which can be time-consuming. This can make it difficult to find the best parameters, which can be subjective in many cases.

Further study of the relationship between sparsity and the number of segments would be fruitful area for further research. Currently, there is nothing linking common segments between different segmentations initialized with different parameters. It could be very enlightening to find a way to track and visualize how particular segments grow and shrink as the regularizing shrinkage parameter changes, and as certain features are or are not selected to remain in the model.

7.2 Conclusions about *Cardinal*

Cardinal is a general, flexible, open-source tool for the statistical analysis of MS imaging experiments. It can be used by researchers with and without background in R and computing. For experimentalists, *Cardinal* provides a full toolchain for multiple workflows, with emphasis on multivariate statistical modeling, inference, and model-based visualization. For developers, *Cardinal* provides a foundation for designing and implementing new methods of computational and statistical analysis of MS imaging experiments.

Cardinal has been well-received by the MS imaging community with over 3,000 downloads, and has an active Google help group where users frequently request new features. These requests have been from both experimentalists, who request new analytic methods, and from statisticians and other developers, who request new developer features.

A major limitation for *Cardinal* has been adapting to the fast-paced development of MS imaging technology and increasing mass and spatial resolution. At release, *Cardinal* was limited to available memory. Rudimentaly support for larger-than-memory data-on-disk was added in v1.3.0, but functionality for such datasets remained limited. While *matter* promises to alleviate some of this difficulty, the statistical methods in *Cardinal* – including spatial shrunken centroids – utilize C and C++ code which assume in-memory matrices. These must be adapted to use larger-than-memory matrix implementations such as *matter* and *bigmemory*.

Support for parallel processing is also frequently requested, but there are still major challenges to solve in terms of how best to implement it – particularly given the growing size of datasets and the complex structure of MS imaging experiments.

7.3 Conclusions about *matter*

matter is a flexible, open-source framework for rapid prototyping with data on disk. It enables development of statistical methods for larger-than-memory datasets.

It is easily adaptable to domain-specific file formats, such as imzML and Analyze 7.5 for MS imaging experiments, without the need for additional file conversion. It provides bounded-memory statistical summeries with data-on-disk. It also provides bounded-memory linear regression for data-on-disk using the *biglm* package. *matter* 's C++ API offers buffered iteration over on-disk vectors and matrices. As a backend for *Cardinal* , it allows processing, visualization, and statistical analysis of larger-than-memory MS imaging experiments which could not be analyzed before.

However, there is a lot of room for improvement. Currently, *matter* only supports dense matrices on disk. Support for sparse matrices on disk is a necessity, particularly for domains such as MS imaging, which relies on the imzML format. Currently *matter* can only be used with the "continuous" imzML format, but not the "processed" imzML format, which must be treated as a kind of sparse matrix. This poses a unique challenge, since the representation on disk is already compressed, so it's uncertain how to best represent the matrix in memory without using a data structure as large as its size on disk.

7.4 General Conclusions

In conclusion, MS imaging is a rapidly advancing field where the experimental technology continues to outpace statistical and computational methodology. Simply loading data and preparing it for statistical analysis can often be difficult. The barriers to entry for statisticians remain high, despite *Cardinal* significantly lowering them. The development of *Cardinal* was necessary for the development of spatial shrunken centroids and other statistical methods. However, as dataset sizes continue to grow, *matter* is quickly becoming a practical necessity for the development of new statistical methods that can handle the new generation of high-resolution, high-throughput MS imaging experiments, which continue to bring new challenges, as biology becomes ever-more-complicated as it approaches the scale of single cells. MS imaging is posing a new set of challenges which combine the statistical complexities of bioinformatics with the computational complexities of big data, where "embarrasingly parallel" is no longer an adequate answer to the size of a dataset when faced with its complex correlation structures. Development of statistically-focused computational infrastructure alongside new statistical methods is the only way forward.

VITA

VITA

Kylie Ariel Bemis was born Kyle Dwayne in 1989 in Indianapolis, IN. She received a B.S. in Statistics and Mathematics from Purdue University in 2010. She received a M.S. in Applied Statistics from Purdue University in 2011. She is an enrolled member of the Zuni tribe. Her hobbies include writing fiction and poetry.

REFERENCES

REFERENCES

[1] J. D. Watrous, T. Alexandrov, and P. C. Dorrestein, “The evolving field of imaging mass spectrometry and its impact on future biological research,” *Journal of Mass Spectrometry*, vol. 46, p. 209, 2011.

[2] A. L. Dill, L. S. Eberlin, C. Zheng, A. B. Costa, D. R. Ifa, L. Cheng, T. A. Masterson, M. O. Koch, O. Vitek, and R. G. Cooks, “Multivariate statistical differentiation of renal cell carcinomas based on lipidomic analysis by ambient ionization imaging mass spectrometry,” *Analytical and Bioanalytical Chemistry*, vol. 398, pp. 2969–2978, October 2010.

[3] A. L. Dill, L. S. Eberlin, A. B. Costa, C. Zheng, D. R. Ifa, L. Cheng, T. A. Masterson, M. O. Koch, O. Vitek, and R. G. Cooks, “Multivariate Statistical Identification of Human Bladder Carcinomas Using Ambient Ionization Imaging Mass Spectrometry,” *Chemistry - A European Journal*, vol. 17, no. 10, pp. 2897–2902, January 2011.

[4] K. D. Bemis, A. Harry, L. S. Eberlin, C. R. Ferreira, S. M. van de Ven, P. Mallick, M. Stolowitz, and O. Vitek, “Probabilistic Segmentation of Mass Spectrometry (MS) Images Helps Select Important Ions and Characterize Confidence in the Resulting Segments,” *Molecular & Cellular Proteomics*, vol. 15, no. 5, pp. 1761–1772, May 2016.

[5] R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu, “Diagnosis of multiple cancer types by shrunken centroids of gene expression,” *Proceedings of the National Academy of Sciences*, vol. 99, pp. 1–6, May 2002.

[6] ——, “Class prediction by nearest shrunken with applications to DNA microarrays,” *Statistical Science*, vol. 18, p. 104, June 2003.

[7] T. Alexandrov and J. H. Kobarg, “Efficient spatial segmentation of large imaging mass spectrometry datasets with spatially aware clustering,” *Bioinformatics*, vol. 27, p. i230, June 2011.

[8] K. D. Bemis, A. Harry, L. S. Eberlin, C. Ferreira, S. M. van de Ven, P. Mallick, M. Stolowitz, and O. Vitek, “Cardinal: an R package for statistical analysis of mass spectrometry-based imaging experiments,” *Bioinformatics*, 2015.

[9] T. Schramm, A. Hester, I. Klinkert, J. P. Both, R. M. Heeren, A. Brunelle, O. Laprévote, N. Desbenoit, F. Robbe M, M. Stoeckli, B. Spengler, and A. Römpp, “imzML – A common data format for the flexible exchange and processing of mass spectrometry imaging data,” *Journal of Proteomics*, vol. 75, p. 5106, 2012.

[10] J. M. Wiseman and B. C. Laughlin, “Desorption Electrospray Ionization (DESI) Mass Spectrometry: A brief introduction and overview,” *Current Separations and Drug Development*, vol. 22, no. 1, pp. 11–14, April 2007.

[11] T. Alexandrov, "MALDI imaging mass spectrometry: statistical data analysis and current computational challenges," *BMC Bioinformatics*, vol. 13, no. Suppl 16, p. S11, November 2012.

[12] C. Yang, Z. He, and W. Yu, "Comparison of public peak detection algorithms for MALDI mass spectrometry data analysis," *BMC Bioinformatics*, vol. 10, 2009.

[13] T. Alexandrov, S. Meding, D. Trede, J. H. Kobarg, B. Balluff, A. Walch, H. Thiele, and P. Maass, "Super-resolution segmentation of imaging mass spectrometry data: Solving the issue of low lateral resolution," *Journal of Proteomics*, vol. 75, no. 1, pp. 237–245, Dec. 2011.

[14] D. Mantini, F. Petrucci, D. Pieragostino, P. Del Boccio, M. Di Nicola, C. Di Ilio, G. Federici, P. Sacchetta, S. Comani, and A. Urbani, "LIMPIC: a computational method for the separation of protein MALDI-TOF-MS signals from noise," *BMC Bioinformatics*, vol. 8, p. 101, 2007.

[15] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images," in *Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay India*, Apr. 1998, pp. 1–8.

[16] K. A. Veselkov, R. Mirnezami, N. Strittmatter, R. D. Goldin, J. Kinross, A. V. M. Speller, T. Abramov, E. A. Jones, A. Darzi, E. Holmes, J. K. Nicholson, and Z. Takats, "Chemo-informatic strategy for imaging mass spectrometry-based hyperspectral profiling of lipid signatures in colorectal cancer," *Proceedings of the National Academy of Sciences*, vol. 111, no. 3, pp. 1216–1221, Jan. 2014.

[17] N. E. Mascini, G. B. Eijkel, P. ter Brugge, J. Jonkers, J. Wesseling, and R. M. A. Heeren, "The Use of Mass Spectrometry Imaging to Predict Treatment Response of Patient-Derived Xenograft Models of Triple-Negative Breast Cancer," *Journal of Proteome Research*, vol. 14, no. 2, pp. 1069–1075, Feb. 2015.

[18] L. S. Ferguson, F. Wulfert, R. Wolstenholme, J. M. Fonville, M. R. Clench, V. A. Carolan, and S. Francese, "Direct detection of peptides and small proteins in fingermarks and determination of sex by MALDI mass spectrometry profiling," *The Analyst*, vol. 137, no. 20, pp. 4686–7, 2012.

[19] C. Wu, A. L. Dill, L. S. Eberlin, R. G. Cooks, and D. R. Ifa, "Mass spectrometry imaging under ambient conditions," *Mass Spectrometry Reviews*, vol. 32, no. 3, pp. 218–243, Sep. 2012.

[20] S. Sarkari, C. Kaddi, R. Bennett, F. Fernandez, and M. Wang, "Comparison of Clustering Pipelines for the Analysis of Mass Spectrometry Imaging Data," pp. 1–4, Jun. 2014.

[21] G. McCombie, D. Staab, M. Stoeckli, and R. Knochenmuss, "Spatial and Spectral Correlations in MALDI Mass Spectrometry Images by Clustering and Multivariate Analysis," *Analytical Chemistry*, vol. 77, no. 19, pp. 6118–6124, Oct. 2005.

[22] S.-O. Deininger, M. P. Ebert, A. Fütterer, M. Gerhard, and C. Röcken, "MALDI Imaging Combined with Hierarchical Clustering as a New Tool for the Interpretation of Complex Human Cancers," *Journal of Proteome Research*, vol. 7, no. 12, pp. 5230–5236, Dec. 2008.

[23] L. S. Eberlin, R. J. Tibshirani, J. Zhang, T. A. Longacre, G. J. Berry, D. B. Bingham, J. A. Norton, R. N. Zare, and G. A. Poultsides, "Molecular assessment of surgical-resection margins of gastric cancer by mass-spectrometric imaging," *Proceedings of the National Academy of Sciences*, vol. 111, no. 7, pp. 2436–2441, Feb. 2014.

[24] T. Alexandrov and A. Bartels, "Testing for presence of known and unknown molecules in imaging mass spectrometry," *Bioinformatics*, vol. 29, no. 18, pp. 2335–2342, September 2013.

[25] C. D. Wijetunge, I. Saeed, B. A. Boughton, J. M. Spraggins, R. M. Caprioli, A. Bacic, U. Roessner, and S. K. Halgamuge, "EXIMS: an improved data analysis pipeline based on a new peak picking method for EXploring Imaging Mass Spectrometry data," *Bioinformatics*, pp. 1–9, June 2015.

[26] T. Alexandrov, M. Becker, S.-O. Deininger, G. Ernst, L. Wehder, M. Grasmair, F. von Eggeling, H. Thiele, and P. Maass, "Spatial Segmentation of Imaging Mass Spectrometry Data with Edge-Preserving Image Denoising and Clustering," *Journal of Proteome Research*, vol. 9, no. 12, pp. 6535–6546, December 2010.

[27] D. Sarkar, *Lattice: Multivariate Data Visualization with R.* New York: Springer, 2008, iSBN 978-0-387-75968-5. [Online]. Available: http://lmdvr.r-forge.r-project.org

[28] M. J. Kane, J. Emerson, and S. Weston, "Scalable strategies for computing with massive data," *Journal of Statistical Software*, vol. 55, no. 14, pp. 1–19, 2013. [Online]. Available: http://www.jstatsoft.org/v55/i14/

[29] D. Adler, C. GlÃd'ser, O. Nenadic, J. OehlschlÃd'gel, and W. Zucchini, *ff: memory-efficient storage of large data on disk and fast access functions*, 2014, r package version 2.2-13. [Online]. Available: https://CRAN.R-project.org/package=ff

[30] B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," *Technometrics*, vol. 4, no. 3, pp. 1–3, Aug. 1962.

[31] T. Lumley, *biglm: bounded memory linear and generalized linear models*, 2013, r package version 0.9-1. [Online]. Available: https://CRAN.R-project.org/package=biglm

[32] D. Calvetti, L. Reichel, and D. C. Sorensen, "An implicitly restarted lanczos method for large symmetric eigenvalue problems," *Electronic Transactions on Numerical Analysis*, vol. 2, pp. 1–21, Mar. 1994.

[33] J. Baglama and L. Reichel, *irlba: Fast Truncated SVD, PCA and Symmetric Eigendecomposition for Large Dense and Sparse Matrices*, 2015, r package version 2.0.0. [Online]. Available: https://CRAN.R-project.org/package=irlba

[34] J. Oetjen, K. Veselkov, J. Watrous, J. S. McKenzie, M. Becker, L. Hauberg-Lotte, J. H. Kobarg, N. Strittmatter, A. K. Mróz, F. Hoffmann, D. Trede, A. Palmer, S. Schiffler, K. Steinhorst, M. Aichler, R. Goldin, O. Guntinas-Lichius, F. von Eggeling, H. Thiele, K. Maedler, A. Walch, P. Maass, P. C. Dorrestein, Z. Takats, and T. Alexandrov, "Benchmark datasets for 3D MALDI- and DESI-imaging mass spectrometry," *GigaScience*, vol. 4, no. 1, pp. 2105–8, May 2015.