Cardinal design and development

Kylie A. Bemis

April 24, 2017

Contents

1 Introduction 1

2 Design overview 2

3 iSet: high-throughput imaging experiments 2
   3.1 SImageSet: pixel-sparse imaging experiments 3
   3.2 MSImageSet: mass spectrometry-based imaging experiments 3

4 ImageData: high-throughput image data 4
   4.1 SImageData: pixel-sparse imaging experiments 4
   4.2 MSImageData: mass spectrometry imaging data 5
      4.2.1 Hashmat: compressed-sparse column matrices 6

5 IAnnotatedDataFrame: pixel metadata for imaging experiments 6

6 MIAPE-Imaging: Minimum Information About a Proteomics Experiment for MS imaging 7

7 MSImageProcess: mass spectral pre-processing information 8

8 ResultSet: analysis results for imaging experiments 8

9 Visualization for high-throughput imaging experiments 8
   9.1 SImageData and MSImageData 9
   9.2 ResultSet 9

10 Testing during development 10
   10.1 Simulating mass spectra 10
   10.2 Timing and diagnostics 10

11 Session info 10

1 Introduction

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.
As an R package, Cardinal allows for the rapid prototyping of new analysis methods. This vignette describes the design of Cardinal data structures for developers interested in writing new R packages using or extending them.

## 2 Design overview

The `iSet` object is the foundational data structure of Cardinal. What is an `iSet`?

- Similar to `eSet` in `Biobase` and `pSet` in `MSnbase`.
- Coordinates high-throughput imaging data, feature data, pixel data, and 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.

## 3 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.

```r
> getClass("iSet")

Virtual Class "iSet" [package "Cardinal"]

Slots:

  Name:          imageData   pixelData   featureData   experimentData
  Class:         ImageData  IAnnotatedDataFrame AnnotatedDataFrame MIAxE

  Name:   protocolData    .__classVersion__
  Class:  AnnotatedDataFrame  Versions

Extends:
Class "VersionedBiobase", directly
Class "Versioned", by class "VersionedBiobase", distance 2
```

Known Subclasses:
Cardinal design and development

Class "SImageSet", directly
Class "ResultSet", directly
Class "MSImageSet", by class "SImageSet", distance 2
Class "CrossValidated", by class "ResultSet", distance 2
Class "PCA", by class "ResultSet", distance 2
Class "PLS", by class "ResultSet", distance 2
Class "OPLS", by class "ResultSet", distance 2
Class "SpatialKMeans", by class "ResultSet", distance 2
Class "SpatialShrunkenCentroids", by class "ResultSet", distance 2

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 4, and the pixelData slot, which will be discussed further in Section 5.

3.1 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.

> getClass("SImageSet")

Class "SImageSet" [package "Cardinal"]

Slots:
Name: imageData pixelData featureData experimentData
Class: SImageData IAnnotatedDataFrame AnnotatedDataFrame MIAxE

Name: protocolData .__classVersion__
Class: AnnotatedDataFrame Versions

Extends:
Class "iSet", directly
Class "VersionedBiobase", by class "iSet", distance 2
Class "Versioned", by class "iSet", distance 3

Known Subclasses: "MSImageSet"

3.2 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.

> getClass("MSImageSet")

Class "MSImageSet" [package "Cardinal"]

Slots:
4.1 **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.

```r
> getClass("SImageData")
```

Class "SImageData" [package "Cardinal"]

Slots:

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>data.frame</td>
</tr>
<tr>
<td>positionArray</td>
<td>array</td>
</tr>
<tr>
<td>dim</td>
<td>numeric</td>
</tr>
<tr>
<td>dimnames</td>
<td>list</td>
</tr>
<tr>
<td>data</td>
<td>environment</td>
</tr>
</tbody>
</table>

Name: storageMode .__classVersion__

Class: character Versions

Extends: Class "ImageData", directly

Class "Versioned", by class "ImageData", distance 2

Known Subclasses: "MSImageData"

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.

### 4.2 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.)

```r
> getClass("MSImageData")
```

Class "MSImageData" [package "Cardinal"]

Slots:

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>data.frame</td>
</tr>
<tr>
<td>positionArray</td>
<td>array</td>
</tr>
<tr>
<td>dim</td>
<td>numeric</td>
</tr>
<tr>
<td>dimnames</td>
<td>list</td>
</tr>
<tr>
<td>data</td>
<td>environment</td>
</tr>
</tbody>
</table>

Name: storageMode .__classVersion__
Class: character Versions

Extends:
Class "SImageData", directly
Class "ImageData", by class "SImageData", distance 2
Class "Versioned", by class "SImageData", distance 3

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

4.2.1 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.

```r
> getClass("Hashmat")
Class "Hashmat" [package "Cardinal"]
```

Slots:

Name: data keys dim dimnames .__classVersion__
Class: list character numeric list Versions

Extends: "Versioned"

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.

5 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.

```r
> getClass("IAnnotatedDataFrame")
```
Class "IAnnotatedDataFrame" [package "Cardinal"]

Slots:

Name: varMetadata data dimLabels __classVersion__
Class: data.frame data.frame character Versions

Extends:
Class "AnnotatedDataFrame", directly
Class "Versioned", by class "AnnotatedDataFrame", distance 2

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".

6 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 Proteomics Experiment for Imaging. Most of its unique slots are based on the imzML specification.

> getClass("MIAPE-Imaging")
Class "MIAPE-Imaging" [package "Cardinal"]

Slots:

Name: title abstract url pubMedIds preprocessing
Class: character character character character list

Name: other name lab contact samples
Class: list character character character list

Name: specimenOrigin specimenType stainingMethod tissueThickness tissueWash
Class: character character character numeric character

Name: embeddingMethod inSituChemistry matrixApplication pixelSize instrumentModel
Class: character character character numeric character

Name: instrumentVendor massAnalyzerType ionizationType scanPolarity softwareName
Class: character character character character character

Name: softwareVersion scanType scanPattern scanDirection lineScanDirection
Class: character character character character character

Name: imageShape __classVersion__
Class: character Versions

Extends:
Class "MIAxE", directly
Class "Versioned", by class "MIAxE", distance 2
7 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.

> getClass("MSImageProcess")

Class "MSImageProcess" [package "Cardinal"]

Slots:

Name: files normalization smoothing baselineReduction
Class: character character character character

Name: spectrumRepresentation peakPicking centroided history
Class: character character logical list

Name: CardinalVersion .__classVersion__
Class: character Versions

Extends: "Versioned"

8 ResultSet: analysis results for imaging experiments

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

> getClass("ResultSet")

Virtual Class "ResultSet" [package "Cardinal"]

Slots:

Name: resultData modelData imageData pixelData
Class: list AnnotatedDataFrame ImageData IAnnotatedDataFrame

Name: featureData experimentData protocolData .__classVersion__
Class: AnnotatedDataFrame MIaxE AnnotatedDataFrame Versions

Extends:
Class "iSet", directly
Class "VersionedBiobase", by class "iSet", distance 2
Class "Versioned", by class "iSet", distance 3


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.

9 Visualization for high-throughput imaging experiments

Cardinal provides a thorough methods for data visualization inspired by the lattice graphics system. Cardinal can display multiple images or plots in a grid of panels based on conditions.
For example, for mass spectrometry imaging, multiple ion images or mass spectra can be plotted together on the same intensity scale. They can be plotted according to different conditions, such as the mean spectra for different phenotypes, etc.

### 9.1 SImageData and MSImageData

The main *Cardinal* walkthrough vignette describes in detail the plot and image methods for SImageData and MSImageData objects, which use *lattice*-style formulae and arguments.

### 9.2 ResultSet

Of interest to developers is writing simple methods for the plotting of ResultSet objects. The plot and image methods for ResultSet make it straightforward to write visualization methods for any kind of analysis results.

The plot method can create plots of results against features (such as model coefficients), while image creates images of results (such as predicted values).

For example, consider the plot and image methods for the PCA class, which is a subclass of ResultSet for principal components analysis.

```r
> selectMethod("plot", c("PCA", "missing"))
Method Definition:

function (x, y, ...) {
  .local <- function (x, formula = substitute(mode ~ mz), mode = "loadings",
    type = "h", ...)
  {
    mode <- match.arg(mode)
    callNextMethod(x, formula = formula, type = type, ...)
  }
  .local(x, ...)
}
<environment: namespace:Cardinal>

Signatures:
  x  y
target "PCA" "missing"
declared "PCA" "missing"

> selectMethod("image", "PCA")
Method Definition:

function (x, ...) {
  .local <- function (x, formula = substitute(mode ~ x * y),
    mode = "scores", ...)
  {
    mode <- match.arg(mode)
    callNextMethod(x, formula = formula, ...)
  }
  .local(x, ...)
}
<environment: namespace:Cardinal>

Signatures:
  x
target "PCA"
defined "PCA"

The left-hand side of the formula (which can be changed by the "mode" argument in the above example) should be an element in the resultData of the ResultSet class. So plot will plot the PC loadings, while image will plot an image of the PC scores.

Such a method will work for two types of results: matrices with the same number of rows as the number of features (for plot), and matrices with the same number of rows as the number of pixels (for image).

Usual lattice-style arguments will work for ResultSet as they would for SImageData and MSImageData, such as "superpose" for plotting results from different models on the same panel or separate panels.

10 Testing during development

Cardinal provides some simple tools to aid in the development of new analysis methods, such as for testing simulated data and timing analyses.

10.1 Simulating mass spectra

The main Cardinal walkthrough vignette describes in detail the generateSpectrum and generateImage methods for generating mass spectra and images.

10.2 Timing and diagnostics

Cardinal provides an option for automatically timing all of its own pre-processing and analysis routines.

> options(Cardinal.timing=TRUE)

Some of its analysis methods such as spatialKMeans and spatialShrunkenCentroids also report timings as part of their standard results.

11 Session info

- R version 3.4.0 (2017-04-21), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C,
  LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C,
  LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Running under: Ubuntu 16.04.2 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.36.0, BiocGenerics 0.22.0, Cardinal 1.8.0, DBI 0.6-1, ProtGenerics 1.8.0,
  biglm 0.9-1, matter 1.2.0
- Loaded via a namespace (and not attached): BiocStyle 2.4.0, MASS 7.3-47, Matrix 1.2-9, Rcpp 0.12.10,
  backports 1.0.5, compiler 3.4.0, digest 0.6.12, evaluate 0.10, grid 3.4.0, htmltools 0.3.5, irlba 2.1.2,
  knitr 1.15.1, lattice 0.20-35, magrittr 1.5, rmarkdown 1.4, rprojroot 1.2, signal 0.7-6, sp 1.2-4, stats4 3.4.0,
  stringi 1.1.5, stringr 1.2.0, tools 3.4.0, yaml 2.1.14