

Interfacing R with Web Technologies for
Data Acquisition and Interactive
Visualization

Carson Sievert

TABLE OF CONTENTS

| | |
|--|-----------|
| LIST OF TABLES | iv |
| LIST OF FIGURES | vi |
| ACKNOWLEDGEMENTS | ix |
| CHAPTER 1 Literature Review | 1 |
| 1.1 What makes a good software interface? | 1 |
| 1.2 Acquiring and wrangling web content in R | 3 |
| 1.2.1 Interfaces for working with web content | 3 |
| 1.2.2 Interfaces for acquiring data on the web | 5 |
| 1.3 Dynamic interactive statistical web graphics | 7 |
| 1.3.1 Why interactive? | 7 |
| 1.3.2 Indirect versus direct manipulation | 10 |
| 1.3.3 Multiple linked views | 11 |
| 1.3.4 Web graphics | 12 |
| 1.3.5 Translating R graphics to the web | 13 |
| 1.3.6 R interfaces for interactive web graphics | 15 |
| CHAPTER 2 Taming PITCHf/x Data with XML2R and pitchRx | 21 |
| ABSTRACT | 22 |
| 2.1 Introduction | 22 |
| 2.1.1 What is PITCHf/x? | 22 |

| | | |
|---|--|-----------|
| 2.1.2 | Why is PITCHf/x important? | 23 |
| 2.1.3 | PITCHf/x applications | 23 |
| 2.1.4 | Contributions of pitchRx and XML2R | 24 |
| 2.2 | Getting familiar with Gameday | 25 |
| 2.3 | Introducing XML2R | 28 |
| 2.3.1 | Constructing file names | 28 |
| 2.3.2 | Extracting observations | 29 |
| 2.3.3 | Renaming observations | 31 |
| 2.3.4 | Linking observations | 33 |
| 2.3.5 | Collapsing observations | 34 |
| 2.4 | Collecting Gameday data with pitchRx | 35 |
| 2.5 | Storing and querying Gameday data | 36 |
| 2.6 | Visualizing PITCHf/x | 39 |
| 2.6.1 | Strike-zone plots and umpire bias | 39 |
| 2.6.2 | 2D animation | 48 |
| 2.6.3 | Interactive 3D graphics | 51 |
| 2.7 | Conclusion | 54 |
| CHAPTER 3 LDAvis: A method for visualizing and interpreting topics | | 55 |
| ABSTRACT | | 56 |
| 3.1 | Introduction | 56 |
| 3.2 | Related Work | 59 |
| 3.2.1 | Topic Interpretation and Coherence | 59 |
| 3.2.2 | Topic Model Visualization Systems | 61 |
| 3.3 | Relevance of tokens to topics | 62 |
| 3.3.1 | Definition of Relevance | 63 |
| 3.3.2 | User Study | 65 |

| | | |
|-----|------------------------------------|----|
| 3.4 | Our Visualization System | 69 |
| 3.5 | Discussion | 72 |

LIST OF TABLES

| | | |
|-----------|--|----|
| Table 2.1 | Structure of PITCHf/x and related Gameday data sources accessible to ‘scrape()’ | 27 |
|-----------|--|----|

LIST OF FIGURES

| | | |
|------------|---|----|
| Figure 1.1 | A demonstration of interactive and dynamic techniques for visualizing high-dimensional relationships in data using the R package <code>tourbrush</code> . You can view this movie online at https://vimeo.com/148050343 or via the supplementary materials | 8 |
| Figure 1.2 | A basic visual depiction of the different approaches to implementing a data pipeline for interactive web graphics. The R packages <code>ggvis</code> and <code>shiny</code> expose the pipeline to users in R, which requires a web server for viewing. The R package <code>crosstalk</code> will allow developers to implement and expose the pipeline on both the server and client levels. | 16 |
| Figure 2.1 | Table relations between Gameday data accessible via <code>'scrape()'</code> . The direction of the arrows indicate a one to possibly many relationship. | 27 |
| Figure 2.2 | Density of called strikes for right-handed batters and left-handed batters (from 2008 to 2013). | 42 |
| Figure 2.3 | Density of called strikes minus density of balls for both right-handed batters and left-handed batters (from 2008 to 2013). The blue region indicates a higher frequency of called strikes and the red region indicates a higher frequency of balls. | 43 |

| | | |
|------------|---|----|
| Figure 2.4 | Probability that a right-handed away pitcher receives a called strike (provided the umpire has to make a decision). Plots are faceted by the handedness of the batter. | 46 |
| Figure 2.5 | Difference between home and away pitchers in the probability of a strike (provided the umpire has to make a decision). The blue regions indicate a higher probability of a strike for home pitchers and red regions indicate a higher probability of a strike for away pitchers. Plots are faceted by the handedness of both the pitcher and the batter. | 47 |
| Figure 2.6 | The last frame of an animation of every four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. The actual animation can be viewed at http://cpsievert.github.io/pitchRx/ani1 . Pitches are faceted by pitcher and batting stance. For instance, the top left plot portrays pitches thrown by Rivera to left-handed batters. . . | 50 |
| Figure 2.7 | The last frame of an animation of averaged four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. The actual animation can be viewed at http://cpsievert.github.io/pitchRx/ani2 . PITCHf/x parameters are averaged over pitch type, pitcher and batting stance. For instance, the bottom right plot portrays an average four-seam and average cutter thrown by Hughes to right-handed batters. | 52 |

| | | |
|------------|--|----|
| Figure 2.8 | 3D scatterplot of pitches from Rivera. Pitches are plotted every one-hundredth of a second. Cutting fastballs are shown in red and four-seam fastballs are shown in blue. The left hand plot takes a viewpoint of Rivera and the right hand plot takes a viewpoint near the umpire. Note these are static pictures of an interactive object. | 53 |
| Figure 3.1 | The layout of LDAvis, with the global topic view on the left, and the token barcharts on the right. Linked selections allow users to reveal aspects of the topic-token relationships compactly. | 58 |
| Figure 3.2 | Dotted lines separating the top-10 most relevant tokens for different values of λ , with the most relevant tokens for $\lambda = 2/3$ displayed and highlighted in green. | 64 |
| Figure 3.3 | A plot of the proportion of correct responses in a user study vs. the value of λ used to compute the most relevant tokens for each topic. | 67 |
| Figure 3.4 | The user has chosen to segment the topics into four clusters, and has selected the green cluster to populate the barchart with the most relevant tokens for that cluster. Then, the user hovered over the ninth bar from the top, 'file', to display the conditional distribution over topics for this token. | 70 |

ACKNOWLEDGEMENTS

This thesis would not be possible without many people. First and foremost, special thanks to my major professor Heike Hofmann. I would not made it to this point without such a warm and friendly mentor who was often more confident in my abilities than I was of my own. Thank you for always supporting me no matter how many things I had going on to distract me from research. I aspire to inherit the same empathy and support that you show to your students on a daily basis.

Another special thanks goes to Di Cook. In the fourth year of my PhD, I was experiencing burnout (and growing tired of Ames), when Di took me to lunch, told me she was transferring to Monash University in Australia, and invited me to join her. Of course, I said yes, and when I arrived, I immediately felt welcomed and a part of the group – all thanks to Di. She strategically assigned me to assist her students with their thesis projects, run R workshops for the university, and most importantly, work on my tennis game. Through this "work" I met so many amazing people and had many memorable experiences. Those 6 months gave me a new perspective on life in general and I am forever grateful for being blessed enough to take the opportunity.

Thank you to many of Heike and Di's former students who came before me (just to name a few: Hadley Wickham, Michael Lawrence, Yihui Xie, Xiaoyue Cheng, Barrett Schloerke, Susan VanderPlas). Your work has not only inspired and enabled my work, but it has also enabled an entire community of people working with data to do amazing things. Without this strong history and community at Iowa State, I would not have had

the courage or the vision to follow such a "non-traditional" research path. I hope the University continues to value this type of work as it teaches students skills that are in high demand and generally improves the way data-driven research is performed.

Thank you to all my collaborators, especially Toby Dylan Hocking. Toby and Susan VanderPlas laid the initial framework for **animint** – which I first worked on as a Google Summer of Code student under Toby's guidance. Toby later went on write the initial version of the `ggplotly()` function in **plotly**, borrowing a lot of ideas from **animint**. As Toby became busy with other things, he introduced me to the plotly team, and eventually handed over the reigns on the project, which has helped to financially support the last year or so of grad school.

Thank you also to the plotly team, and in particular, the software engineers who work on the open source project plotly.js. My work has benefited greatly from your responsiveness to my questions, feature requests, and bug reports. I have a great amount of respect for the work that you do, and I hope this project keeps improving at its current break neck pace.

Finally, thank you to my family for their encouragement and keeping me grounded throughout this experience. Thank you to my father for the initial encouragement to pursue a PhD, conversations surrounding work-life balance, and also pushing me to "graduate before I'm 40". Thank you to my mother for her unconditional love, endless care, and pretending to understand what I do for a living. Thank you to my brothers for providing me with shelter, beer, and Twins tickets. Thank you all for your willingness to drop everything to help me at any given moment. I can't say I've always been as willing, as I have been selfish with my time during my PhD, but I hope to change that after graduation.

1 Literature Review

1.1 What makes a good software interface?

Unwin and Hofmann (Unwin and Hofmann 2009) discuss the strengths, weaknesses, and differences between using graphical and command-line interfaces for data analysis. Graphical user interfaces (GUIs) can be much more intuitive to use, but at the cost of being less flexible, precise, and repeatable. Unwin and Hofmann argue statistical software should strive to achieve a synergy of two that leverages both of their strengths. That is, a command-line interface when we can precisely describe what we want and a graphical interface for “searching for information and interesting structures without fully specified questions.”

Unwin and Hofmann further discuss the different audiences these interfaces attract. Command-line interfaces typically attract “power users” such as applied statisticians and statistical researchers in a university, whereas more casual users of statistical software typically prefer a GUI. In later sections, we discuss GUIs in greater detail within the context of interactive statistical graphics. For now, we briefly discuss some best practices for designing a command-line interface for statistical computing in R.

Before authoring an interface, one should establish the target audience, the class of problems it should address, and loosely define how the interface should actually work. During this process, it may also be helpful to identify your audience as being primarily composed of *software developers* or *data analysts*. Developers are typically more interested

in using the interface to develop novel software or incorporating the functionality into a larger scientific computing environment (Jereon Ooms 2014). In this case, interactive exploration and troubleshooting is not always a luxury, so robust functionality is of utmost importance. On the other hand, analysts interfaces should work well in an interactive environment since this caters to rapid prototyping of ideas and troubleshooting of errors.

Good developer interfaces often make it easier to implement good analyst interfaces. A great recent example of a good developer interface is the R package **Rcpp**, which provides a seamless interface between R with C++ (Eddelbuettel 2013). To date, more than 500 R packages use **Rcpp** to make interfaces that are both expressive and efficient, including the highly influential analyst interfaces such as **tidyr** and **dplyr** (Wickham 2014); (Wickham and Francois 2014). These interfaces help analysts focus on the primary task of wrangling data into a form suitable for visualization and statistical modeling, rather than focusing on the implementation details behind how the transformations are performed. (Donoho 2015) argues that these interfaces “May have more impact on today’s practice of data analysis than many highly-regarded theoretical statistics papers”.

Evaluating statistical computing interfaces is certainly a subjective matter since we all have different tastes, different backgrounds, and have different needs. It seems reasonable to evaluate an interface based on its effectiveness and efficiency in aiding a user complete their task, but as (Unwin and Hofmann 2009) points out, “There is a tendency to judge software by the most powerful tools they provide (whether with a good interface or not)”. As a result, all too often, analysts must spend time gaining the skills of a software developer. Good analyst interfaces often abstract functionality from developer interfaces in a way that allow analysts to focus on their primary task of acquiring/analyzing/modeling/visualizing data, rather than the implementation details. The following focuses on such work with respect to acquiring data from the web and interactive statistical web graphics.

1.2 Acquiring and wrangling web content in R

1.2.1 Interfaces for working with web content

R has a rich history of interfacing with web technologies for accomplishing a variety of tasks such as requesting, manipulating, and creating web content. As an important first step, extending ideas from (Chambers 1999), Brian Ripley implemented the `connections` interface for file-oriented input/output in R (Ripley 2001). This interface supports a variety of common transfer protocols (HTTP, HTTPS, FTP), providing access to most files on the web that can be identified with a Uniform Resource Locator (URL). Connection objects are actually external pointers, meaning that, instead of immediately reading the file, they just point to the file, and make no assumptions about the actual contents of the file.

Many functions in the base R distribution for reading data (e.g., `scan`, `read.table`, `read.csv`, etc.) are built on top of connections, and provide additional functionality for parsing well-structured plain-text into basic R data structures (vector, list, data frame, etc.). However, the base R distribution does not provide functionality for parsing common file formats found on the web (e.g., HTML, XML, JSON). In addition, the standard R connection interface provides no support for communicating with web servers beyond a simple HTTP GET request (Lang 2006).

The **RCurl**, **XML**, and **RJSONIO** packages were major contributions that drastically improved our ability to request, manipulate, and create web content from R (Nolan and Temple Lang 2014). The **RCurl** package provides a suite of high and low level bindings to the C library libcurl, making it possible to transfer files over more network protocols, communicate with web servers (e.g., submit forms, upload files, etc.), process their responses, and handle other details such as redirects and authentication (Temple Lang

2014a). The **XML** package provides low-level bindings to the C library libxml2, making it possible to download, parse, manipulate, and create XML (and HTML) (Lang 2013). To make this possible, **XML** also provides some data structures for representing XML in R. The **RJSONIO** package provides a mapping between R objects and JavaScript Object Notation (JSON) (Temple Lang 2014b). These packages were heavily used for years, but several newer interfaces have made these tasks easier and more efficient.

The **curl**, **httr**, and **jsonlite** packages are more modern R interfaces for requesting content on the web and interacting with web servers. The **curl** package provides a much simpler interface to libcurl that also supports streaming data (useful for transferring large data), and generally has better performance than **RCurl** (Ooms 2015). The **httr** package builds on **curl** and organizes its functionality around HTTP verbs (GET, POST, etc.) (Wickham 2015a). Since most web application programming interfaces (APIs) organize their functionality around these same verbs, it is often very easy to write R bindings to web services with **httr**. The **httr** package also builds on **jsonlite** since it provides consistent mappings between R/JSON and most modern web APIs accept and send messages in JSON format (Jeroen Ooms 2014a). These packages have already had a profound impact on the investment required to interface R with web services, which are useful for many things beyond data acquisition. For example, it is now easy to install R packages hosted on the web (**devtools**), perform cloud computing (**analogsea**), and archive/share computational outputs (**dvn**, **rfigshare**, **RAmazonS3**, **googlesheets**, **rdrop2**, etc.).

The **rvest** package builds on **httr** and makes it easy to manipulate content in HTML/XML files (Wickham 2015c). Using **rvest** in combination with SelectorGadget, it is often possible to extract structured information (e.g., tables, lists, links, etc) from HTML with almost no knowledge/familiarity with web technologies. The **XML2R** package has a similar goal of providing an interface to acquire and manipulate XML content into tabular

R data structures without any working knowledge of XML/XSLT/XPath (Sievert 2014b). As a result, these interfaces reduce the startup costs required for analysts to acquire data from the web.

Packages such as **XML**, **XML2R**, and **rvest** can download and parse the source of web pages, which is *static*, but extracting *dynamic* web content requires additional tools. The R package **rdom** fills this void and makes it easy to render and access the Document Object Model (DOM) using the headless browsing engine phantomjs (Sievert 2015a). The R package **RSelenium** can also render dynamic web pages and simulate user actions, but its broad scope and heavy software requirements make it harder to use and less reliable compared to **rdom** (Harrison 2014). **rdom** is also designed to work seamlessly with **rvest**, so that one may use the `rdom()` function instead of `read_html()` to render, parse, and return the DOM as HTML (instead of just the HTML page source).

Any combination of these R packages may be useful in acquiring data for personal use and/or providing a higher-level interface to specific data source(s) to increase their accessibility. The next section focuses on such interfaces.

1.2.2 Interfaces for acquiring data on the web

The web provides access to the world’s largest repository of publicly available information and data. This provides a nice *potential* resource both teaching and practicing applied statistics, but to be practical useful, it often requires a custom interface to make data more accessible. If publishers follow best practices, a custom interface to the data source usually is not needed, but this is rarely the case. Many times structured data is embedded within larger unstructured documents, making it difficult to incorporate into a data analysis workflow. This is especially true of data used to inform downstream web applications, typically in XML and/or JSON format. There are two main ways to make such data more

accessible: (1) package, document, and distribute the data itself (2) provide functionality to acquire the data.

If the data source is fairly small, somewhat static, and freely available with an open license, then we can directly provide data via R packaging mechanism. In this case, it is best practice for package authors include scripts used to acquire, transform, and clean the data. This model is especially nice for both teaching and providing examples, since users can easily access data by installing the R package. (Wickham 2015b) provides a nice section outlining the details of bundling data with R packages.¹

R packages that just provide functionality to acquire data can be more desirable than bundling it for several reasons. In some cases, it helps avoid legal issues with rehosting copyrighted data. Furthermore, the source code of R packages can always be inspected, so users can verify the cleaning and transformations performed on the data to ensure its integrity, and suggest changes if necessary. They are also versioned, which makes the data acquisition, and thus any downstream analysis, more reproducible and transparent. It is also possible to handle dynamic data with such interfaces, meaning that new data can be acquired without any change to the underlying source code. As explained in Taming PITCHf/x Data with XML2R and pitchRx, this is an important quality of the **pitchRx** R package since new PITCHf/x data is made available on a daily basis.

Perhaps the largest centralized effort in this domain is lead by rOpenSci, a community of R developers that, at the time of writing, maintains more than 50 packages providing access to scientific data ranging from bird sightings, species occurrence, and even text/metadata from academic publications. This provides a tremendous service to researchers who want to spend their time building models and deriving insights from data, rather than learning the programming skills necessary to acquire and clean it.

¹This section is freely available online <http://r-pkgs.had.co.nz/data.html>.

It’s becoming increasingly clear that “meta” packages that standardize the interface to data acquisition/curation in a particular domain would be tremendously useful. However, it is not clear how such interfaces should be designed. The R package **etl** is one step in this direction and aims to provide a standardized interface for *any* data access package that fits into an Extract-Transform-Load paradigm (Baumer and Sievert, n.d.). The package provides generic **extract-transform-load** functions, but requires package authors to write custom **extract-transform** methods for the specific data source. In theory, the default **load** method works for any application; as well as other database management operations such as **update** and **clean**.

1.3 Dynamic interactive statistical web graphics

1.3.1 Why interactive?

Unlike computer graphics which focuses on representing reality, virtually; data visualization is about garnering abstract relationships between multiple variables from visual representation. The dimensionality of data, the number of variables can be anything, usually more than 3D, which summons a need to get beyond 2D canvasses for display. Technology enables this, enabling the user to see many views, query and link components. As demonstrated in Figure 1.1 using the R package **tourbrush** (Sievert 2015b), interactive and dynamic graphics allow us to go beyond the constraints of low-dimensional displays to see high-dimensional relationships in data.

Dynamic interactive statistical graphics is useful for descriptive statistics, and also to help build better inferential models. Any statistician is familiar with diagnosing a model by plotting data in the model space (e.g., residual plot, qqplot). This works well for determining if the assumptions of a model are adequate, but rarely suggests that our

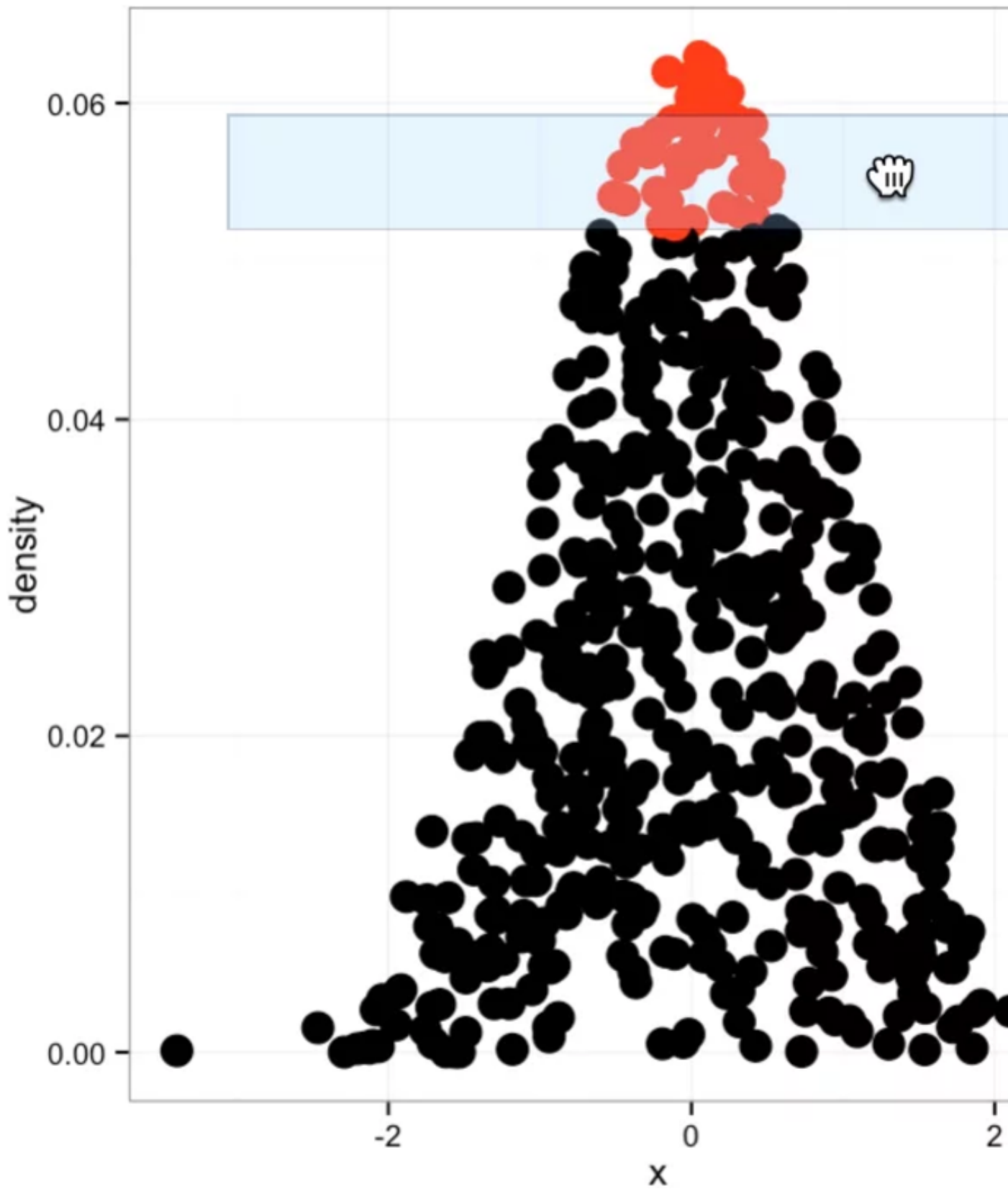


Figure 1.1 A demonstration of interactive and dynamic techniques for visualizing high-dimensional relationships in data using the R package *tourbrush*. You can view this movie online at <https://vimeo.com/148050343> or via the supplementary materials

model neglects important features in the data. To combat this problem, Wickham, Cook, and Hofmann (2015) suggest to plot the model in the data space and use dynamic interactive statistical graphics to do so. Interactive graphics have also proved to be useful for exploratory model analysis, a situation where we have many models to evaluate, compare, and critique (Unwin, Volinsky, and Winkler 2003); (Urbanek 2004); (Ripley 2004); (Unwin 2006); (Wickham 2007). With such power comes responsibility that we can verify that visual discoveries are real, and not due to random chance (Buja et al. 2009); (Majumder, Hofmann, and Cook 2013).

The ASA Section on Statistical Computing and Graphics maintains a video library which captures many useful dynamic interactive statistical graphics techniques. Several videos show how XGobi (predecessor to GGobi), a dynamic interactive statistical graphics system, can be used to reveal high-dimensional relationships and structures that cannot be easily identified using numerical methods alone (Swayne, Cook, and Buja 1998).² Another notable video shows how the interactive graphics system *mondrian* can be used to quickly find interesting patterns in high-dimensional data using exploratory data analysis (EDA) techniques (Theus and Urbanek 2008).³ The most recent video shows how dynamic interactive techniques can help interpret a topic model (a statistical mixture model applied to text data) using the R package **LDavis** (Sievert and Shirley 2014), which is the first web-based visualization in the library, and is discussed at depth in *LDavis: A method for visualizing and interpreting topics*.

In order to be practically useful, interactive statistical graphics must be fast, flexible, accessible, portable, and reproducible. In general, over the last 20-30 years interactive graphics systems were fast and flexible, but were generally not easily accessible, portable, or reproducible. The web browser provides a convenient platform to combat these

²For example, <http://stat-graphics.org/movies/xgobi.html> and <http://stat-graphics.org/movies/grand-tour.html>

³<http://stat-graphics.org/movies/tour-de-france.html>

problems. For example, any visualization created with **LDavis** can be shared through a Uniform Resource Locator (URL), meaning that anyone with a web browser and an internet connection can view and interact with a visualization. Furthermore, we can link anyone to any possible state of the visualization by encoding selections with a URL fragment identifier. This makes it possible to link readers to an interesting state of a visualization from an external document, while still allowing them to independently explore the same visualization and assess conclusions drawn from it.⁴

1.3.2 Indirect versus direct manipulation

Even within the statistical graphics community, the term *interactive* graphics can mean wildly different things to different people (Swayne and Klinke 1999). Some early statistical literature on the topic uses *interactive* in the sense that an interactive command-line prompt allows users to create graphics on-the-fly (R. A. Becker 1984). That is, users enter commands into the command-line prompt, the prompts evaluates the command, and prints the result (known as the read-eval-print loop (REPL)). Modifying a command to generate another variation of a particular result (e.g., to restyle a static plot) can be thought of as a type of interaction that some might call *indirect manipulation*.

Indirect manipulation can be achieved both from the command-line or from a graphical user interface (GUI). Indirect manipulation from the command-line is more flexible since we have complete control over the commands, but it is also more cumbersome since we must translate our thoughts into code. Indirect manipulation via a GUI is more restrictive, but it helps reduces the gulf of execution (i.e., easier to generate desired output) for end-users (Hutchins, Hollan, and Norman 1985). In this sense, a GUI can be useful, even for experienced programmers, when the command-line interface impedes our primary task of deriving insight from data.

⁴A good example of is <http://cpsievert.github.io/LDAvis/reviews/reviews.html>

In many cases, the gulf of execution can be further reduced through direct manipulation. Roughly speaking, within the context of interactive graphics, direct manipulation occurs whenever we interact with a plot and reveal new information tied to the event. Cook and Swayne (2007) use the terms dynamic graphics and direct manipulation to characterize “plots that respond in real time to an analyst’s queries and change dynamically to re-focus, link to information from other sources, and re-organize information.” Directly manipulating multiple linked views to make graphical queries is a very powerful framework for exploring information, and inspires the last 3 chapters of this thesis.

A simple example to help demonstrate the differences between these interactive techniques would be in an analysis of variance (ANOVA) via multiple boxplots. By default, most plotting libraries sort categories alphabetically, but this is usually not optimal for visual comparison of groups. With a static plotting library such as **ggplot2**, we could indirectly manipulate the default by going back to the command-line, reordering the factor levels of the categorical variables, and regenerate the plot (Wickham 2009). This is flexible and precise since we may order the levels by any measure we wish (e.g., Median, Mean, IQR, etc.), but it would be much quicker and easier if we had a GUI with a drop-down menu for most of the reasonable sorting options. In a general purpose interactive graphics system such as *mondrian*, one can use direct manipulation to directly click and drag on the categories to reorder them, making it quick and easy to compare any two groups of interest (Theus and Urbanek 2008).

1.3.3 Multiple linked views

A general purpose interactive statistical graphics system should possess many direct manipulation techniques such as identifying (i.e., mousing over points to reveal labels), focusing (i.e., view size adjustment, pan and zoom), brushing/identifying, etc. However,

it is the intricate management of information across multiple views of data in response to user events that is most valuable. Extending ideas from (Andreas Buja and McDonald 1988), (Wickham et al. 2010) point out that any visualization system with linked views must implement a data pipeline. That is, a “central commander” must be able to handle interaction(s) with a given view, translate its meaning to the data space, and update any linked view(s) accordingly. In order to do so, the commander must know, and be able to compute, function(s) from data to visual space, as well as from visual space to the data. Implementing a pipeline that is fast, general, and able to handle statistical transformations is incredibly difficult. Unfortunately, literature on the implementation of such pipelines is virtually non-existent, but Xie, Hofmann, and Cheng (2014) provides a nice overview of the implementation details in the R package **cranvas** (Yihui Xie 2013).

1.3.4 Web graphics

Thanks to the constant evolution and eventual adoption of HTML5 as a web standard, the modern web browser now provides a viable platform for building an interactive statistical graphics systems. HTML5 refers to a collection of technologies, each designed to perform a certain task, that work together in order to present content in a web browser. The Document Object Model (DOM) is a convention for managing all of these technologies to enable *dynamic* and *interactive* web pages. Among these technologies, there are several that are especially relevant for interactive web graphics:

1. HTML: A markup language for structuring and presenting web content.
2. SVG: A markup language for drawing scalable vector graphics.
3. CSS: A language for specifying styling of web content.
4. JavaScript: A language for manipulating web content.

Juggling all of these technologies just to create a simple statistical plot is a tall order. Thankfully, HTML5 technologies are publicly available, and benefit from thriving community of open source developers and volunteers. In the context of web-based visualization, the most influential contribution is Data Driven Documents (D3), a JavaScript library which provides high-level semantics for binding data to web content (e.g., SVG elements) and orchestrating scene updates/transitions (Heer 2011). D3 is wildly successful because it builds upon web standards, without abstracting them away, which fosters customization and interoperability. However, compared to a statistical graphics environments like R, creating basic charts is complicated, and a large amount of code must be hard-wired to each visualization. Fortunately, there are a number of ways to provide higher-level interfaces to web graphics, and we focus on R interfaces.

1.3.5 Translating R graphics to the web

There are a few ways to simply translate R graphics to a web format, such as SVG. R has built-in support for a SVG graphics device, made available through the `svg()` function, but it can be quite slow, which inspired the new **svglite** package (Wickham et al. 2016). The **SVGAnnotation** package provides some functionality to post-process SVG files generated with `svg()` to add some basic interactivity and animation (Nolan and Temple Lang 2012). The **gridSVG** package is specially designed to translate **grid** graphics (e.g., **ggplot2**, **lattice**, etc.) to SVG, and preserves the naming information of grid objects, making it easier to layer on interactive functionality (Potter and Murrell 2012). Fujino (2015) uses **gridSVG** to enable linked brushing between **ggplot2** graphics, but only implements a few chart types. Riutta et. al. and Russell (2015) uses **gridSVG** to provide pan and zoom capability to virtually any R graphic.

The **animint** and **plotly** packages take a different approach to translating **ggplot2** graphics to a web format (Hocking, VanderPlas, and Sievert 2015); (Sievert et al. 2016). Instead of translating directly to SVG via **gridSVG**, they extract relevant information from the internal representation of a **ggplot2** graphic⁵, store it in JavaScript Object Notation (JSON), and pass the JSON as input to a JavaScript function, which then produces a web based visualization. It is becoming more and more popular to see JavaScript graphing libraries use this design pattern (sometimes referred to as a JSON specification or schema), since it separates out *what* information is contained in the graphic from *how* to actually draw it. This has a number of advantages; for example, **plotly** graphics can be rendered in SVG, or using WebGL (based on HTML5 canvas, not SVG) which allows the browser to render many more graphical marks by leveraging the GPU.

Converting static graphics to web formats such as SVG or canvas not only allows us to embed the graphics into larger HTML documents, but it also allows us to inject basic interactive features at no or little cost to the user. For example, in both **animint** and **plotly**, we provide tool-tips (to obtain data-related information for each graphical mark) and clickable legends that show/hide graphical marks corresponding to the legend entry. In the case of **animint**, we have also extended **ggplot2**'s grammar of graphics implementation to enable animations and categorical linking between plots with relatively small amount of effort by users. This extension is discussed at length in Two new keywords for interactive, animated plot design: `clickSelects` and `showSelected`. In the case of **plotly**, we have also enabled animations, highlighting, and linked highlighting (even between non-plotly graphics). These features are discussed at length in `plotly` for R.

⁵For a visual display of the internal representation used to render a **ggplot2** graph, see my **shiny** app here <https://gallery.shinyapps.io/ggtree>.

1.3.6 R interfaces for interactive web graphics

Translating existing graphics to a web-based format is useful for quickly breathing new life into existing code, but it is fairly limited in how far we can take it. Assuming the goal is to have a general, yet high-level, interface for creating highly dynamic interactive web graphics from R, we're better off building a new interface designed exactly for this purpose. The first serious attempt in this direction was the R package **rCharts**, whose R interface is heavily inspired by **lattice** (Vaidyanathan 2013). The most impressive result of **rCharts**'s design is its ability to interface with many different JavaScript charting libraries. However, **rCharts** has little to no support for coordination of dynamic linked views from R.

Another notable interface for creating interactive web graphics from R is **ggvis**, a reworking of ggplot2's grammar of graphics to incorporate interactivity (Chang and Wickham 2015). Similar to **animint**, **ggvis** encodes plot specific information as JSON, but instead of writing a JavaScript renderer from the ground up, it uses Vega, a popular JSON schema for creating web-based graphics (Heer 2014). This limits the flexibility of **ggvis**, but it also drastically reduces the overhead in maintaining such a software project, allowing the focus to be on building a grammar for expressing interactions from R.

The current version of **ggvis** uses an old version of vega, before a grammar for interactive graphics was added to its JSON schema (Heer 2017). In order to respond to user interactions with vega graphics, **ggvis** has its own custom JavaScript designed specifically for vega. To enable support for coordinated linked views, it exposes the data pipeline to users via the R package **shiny**, a framework for writing web applications in R (Chang et al. 2015). A web application is a website which, when visited by users (aka clients), communicates with a web server. This approach is useful when a website needs to execute code that can not be executed in the web browser (e.g., R code). Figure 1.2 provides

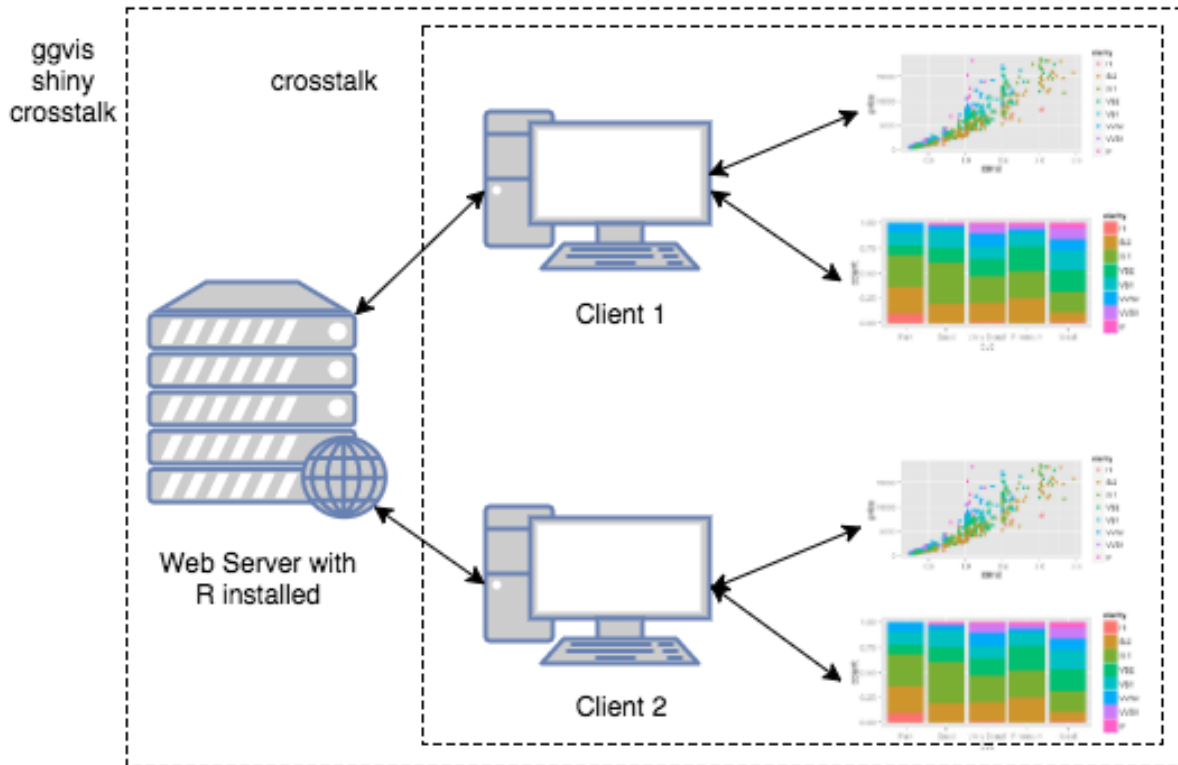


Figure 1.2 A basic visual depiction of the different approaches to implementing a data pipeline for interactive web graphics. The R packages `ggvis` and `shiny` expose the pipeline to users in R, which requires a web server for viewing. The R package `crosstalk` will allow developers to implement and expose the pipeline on both the server and client levels.

a visual demonstration of this model and its relation to the data pipeline necessary for coordinating linked views.

Generally speaking, websites that render entirely client-side are more desirable since they are easier to share, more responsive, and require less computational resources to run⁶. However, the client-server approach can be very useful for dynamically performing statistical computations, a key characteristic of most interactive statistical graphics systems. (Urbanek and Horner 2015) and (Jeroen Ooms 2014b) also allow us to execute R code on a web server, and retrieve output via HTTP, but **shiny** is the most heavily

⁶The <http://www.shinyapps.io/> service helps to provide easy access to a **shiny** server (a web server running special shiny software), so that **shiny** apps can be shared via a URL, for example: <https://hadley.shinyapps.io/14-ggvis/linked-brushing.Rmd>

used since apps can be written entirely in R using a very powerful, yet approachable, reactive programming framework for handling user events. There are also many convenient shortcuts for creating attractive HTML input forms, making it incredibly easy to go from R script to an web app powered by R that dynamically updates when users alter input values. In other words, **shiny** makes it quick and easy to write web-based GUIs with support for indirect manipulation.

Historically, an advanced understanding of **shiny** and JavaScript was required to implement direct manipulation in a **shiny** app. Recently, **shiny** added support for retrieving information on user events with static R graphics⁷, allowing developers to coordinate views in a web app, with no JavaScript involved. This is a powerful tool for R users, but it has its weaknesses. Most importantly, its not clear how to handle interactions when positional scales are categorical (e.g., a bar chart) or how to provide a visual clue that something has been selected.

The touring video in Figure 1.1 purposefully uses **shiny**'s built-in support for brushing to demonstrate the problem with providing a visual clue. This points to the fundamental problem in using non-web-based graphics to implement interactive graphics in a web browser: every time the view updates, the display must be redrawn, resulting in a “glitch” effect. If the plot being brushed used native web graphics (e.g., SVG), it would allow for finer control over how the view updates in response to user interactions and/or dynamic data. On the other hand, since **ggvis** is web-based, and has special client-side functionality, it knows how to smoothly transition from one frame to the next when provided with new data from the **shiny** server, which is crucial for constructing a mental model of the data space. Having richer interfaces for generating web-based interactive graphics from R that can share selections, and handle smooth transitions, would make this, and many other examples, generally better.

⁷This website shows what information is sent from the client to the server when users interact with plot(s) via mouse event(s) – <http://shiny.rstudio.com/gallery/plot-interaction-basic.html>

Many web-based graphing toolkits have appeared since the advent of **rCharts**, making a single package that interfaces with *every* toolkit infeasible. Some ideas deriving from work on **rCharts**, such as providing the glue to render plots in various contexts (e.g., the R console, shiny apps, and **rmarkdown** documents), have evolved into the R package **htmlwidgets** (Vaidyanathan et al. 2015). Having built similar bridges for **animint** and **LDavis**, I personally know and appreciate the amount of time and effort this package saves other package authors.

The **htmlwidgets** framework is not constrained to just graphics, it simply provides a set of conventions for authoring web content from R. Numerous JavaScript data visualization libraries are now made available using this framework, most designed for particular use cases, such as **leaflet** for geo-spatial mapping, **dygraphs** for time-series, and **networkD3** for networks (Cheng and Xie 2015); (Vanderkam and Allaire 2015); (Gandrud, Allaire, and Russell 2015).⁸ There are also HTML widgets that provide an interface to more general purpose visualization JavaScript libraries such as **plotly**, **rbokeh**, and **rcdimple** (Sievert et al. 2016); (Hafen and team 2015); (Kiernander et al. 2015). Most of these JavaScript libraries provide at least some native support for direct manipulation such as identifying (i.e., mousing over points to reveal labels), focusing (i.e., pan and zoom), and sometimes highlighting (i.e., brushing over points to highlight points in another view). More often than not, the support for dynamic and linked views is lacking, especially if we want to define the linking in R, and produce a standalone HTML document.

The R package **crosstalk** is a new framework for coordinating arbitrary HTML widgets (Cheng 2015a). It provides both an R and a JavaScript API for querying selections, meaning **crosstalk** powered HTML widgets can work with or without **shiny**, and if implemented carefully by HTML widget authors, provides a means for coordinating multiple HTML widgets without shiny. Generally speaking, **crosstalk** just provides a

⁸For more examples and information, see <http://www.htmlwidgets.org/> and <http://hafen.github.io/htmlwidgetsgallery/>

standard way to set, store, and access selection values in the browser, so the actual logic for updating views based on the selection value(s) is on the HTML widget author, and this part is far from trivial. In a sense, this project is similar to the work of North and Shneiderman (1999), which provides semantics for “snapping together” arbitrary views that are aware of the relational schema, but does so in a web-based environment, rather than requiring a machine running Windows.

The first HTML widget to leverage **crosstalk** was (Cheng 2015b), but is limited to linked brushing on scatterplots.⁹ Currently, there are a couple other R packages with **crosstalk** support, including **leaflet** and **listviewer**, but **plotly** is the only package which supports a non-identity functions between the data and displays. It also has rich support for interaction types, including mouse hover, click, and multiple types of click+drag selections.

Having HTML widgets that can share selections with each other will be a huge step forward for web-based interactive graphics. With some effort and careful implementation by HTML widget authors, it may be possible to provide sensible defaults for updating views between arbitrary widgets, and users that know some JavaScript will also be able to customize or extend these defaults from R. The **htmlwidget** package provides conventions for this, by allowing one to send arbitrary JavaScript functions from R that execute after the widget has rendered in the browser. The biggest problem in implementing coordinated widgets will be in managing data structures, since each widget will likely have its own data structure for representing a selection. In this case, in order to coordinate them, users may have to embed widgets in a shiny app to access and organize selections. This gives users tremendous control over sharing selections, but may limit control over smooth transitions between states of a given widget (a key characteristic

⁹See, for example, <http://rpubs.com/jcheng/crosstalk-demo>

of dynamic graphics), and increases the amount of complexity involved in sharing their work.

2 Taming PITCHf/x Data with XML2R and pitchRx

This chapter is a paper published in *The R Journal* (Sievert 2014b). I am the sole author of the paper which is available online here <https://journal.r-project.org/archive/2014-1/sievert.pdf>

The formatting of paper has been modified to make for consistent typesetting across the thesis.

ABSTRACT

XML2R is a framework that reduces the effort required to transform XML content into tables in a way that preserves parent to child relationships. **pitchRx** applies **XML2R**'s grammar for XML manipulation to Major League Baseball Advanced Media (MLBAM)'s Gameday data. With **pitchRx**, one can easily obtain and store Gameday data in a remote database. The Gameday website hosts a wealth of XML data, but perhaps most interesting is PITCHf/x. Among other things, PITCHf/x data can be used to recreate a baseball's flight path from a pitcher's hand to home plate. With **pitchRx**, one can easily create animations and interactive 3D scatterplots of the baseball's flight path. PITCHf/x data is also commonly used to generate a static plot of baseball locations at the moment they cross home plate. These plots, sometimes called *strike-zone plots*, can also refer to a plot of event probabilities over the same region. **pitchRx** provides an easy and robust way to generate strike-zone plots using the **ggplot2** package.

2.1 Introduction

2.1.1 What is PITCHf/x?

PITCHf/x is a general term for a system that generates a series of 3D measurements of a baseball's path from a pitcher's hand to home plate (Alt and White 2008).¹ In an attempt to estimate the location of the ball at any time point, a quadratic regression

¹A *pitcher* throws a ball to the opposing *batter*, who stands besides home plate and tries to hit the ball into the field of play.

model with nine parameters (defined by the equations of motion for constant linear acceleration) is fit to each pitch. Studies with access to the actual measurements suggest that this model is quite reasonable – especially for non-knuckleball pitches (Nathan 2008). That is, the fitted path often provides a reasonable estimate (within a couple of inches) of the actual locations. Unfortunately, only the parameter estimates are made available to the public. The website that provides these estimates is maintained by MLBAM and hosts a wealth of other baseball related data used to inform MLB’s Gameday webcast service in near real time.

2.1.2 Why is PITCHf/x important?

On the business side of baseball, using statistical analysis to scout and evaluate players has become mainstream. When PITCHf/x was first introduced, DiMeo (2007) proclaimed it as,

“The new technology that will change statistical analysis [of baseball] forever.”

PITCHf/x has yet to fully deliver this claim, partially due to the difficulty in accessing and deriving insight from the large amount of complex information. By providing better tools to collect and visualize this data, **pitchRx** makes PITCHf/x analysis more accessible to the general public.

2.1.3 PITCHf/x applications

PITCHf/x data is and can be used for many different projects. It can also complement other baseball data sources, which poses an interesting database management problem. Statistical analysis of PITCHf/x data and baseball in general has become so popular that it has helped expose statistical ideas and practice to the general public. If you

have witnessed television broadcasts of MLB games, you know one obvious application of PITCHf/x is locating pitches in the strike-zone as well as recreating flight trajectories, tracking pitch speed, etc. Some on-going statistical research related to PITCHf/x includes: classifying pitch types, predicting pitch sequences, and clustering pitchers with similar tendencies (Pane et al. 2013).

2.1.4 Contributions of **pitchRx** and **XML2R**

The **pitchRx** package has two main focuses (Sievert 2014a). The first focus is to provide easy access to Gameday data. Not only is **pitchRx** helpful for collecting this data in bulk, but it has served as a helpful teaching and research aide (<http://baseballwithr.wordpress.com/> is one such example). Methods for collecting Gameday data existed prior to **pitchRx**; however, these methods are not easily extensible and require juggling technologies that may not be familiar or accessible (Fast 2007). Moreover, these working environments are less desirable than R for data analysis and visualization. Since **pitchRx** is built upon **XML2R**'s united framework, it can be easily modified and/or extended (Sievert 2014c). For this same reason, **pitchRx** serves as a model for building customized XML data collection tools with **XML2R**.

The other main focus of **pitchRx** is to simplify the process creating popular PITCHf/x graphics. Strike-zone plots and animations made via **pitchRx** utilize the extensible **ggplot2** framework as well as various customized options (Wickham 2009). **ggplot2** is a convenient framework for generating strike-zone plots primarily because of its facet schema which allows one to make visual comparisons across any combination of discrete variable(s). Interactive 3D scatterplots are based on the **rgl** package and useful for gaining a new perspective on flight trajectories (Adler, Murdoch, and others, n.d.).

2.2 Getting familiar with Gameday

Gameday data is hosted and made available for free thanks to MLBAM via <http://gd2.mlb.com/components/game/mlb/>.² From this website, one can obtain many different types of data besides PITCHf/x. For example, one can obtain everything from structured media metadata to insider tweets. In fact, this website's purpose is to serve data to various <http://mlb.com> web pages and applications. As a result, some data is redundant and the format may not be optimal for statistical analysis. For these reasons, the `scrape` function is focused on retrieving data that is useful for PITCHf/x analysis and providing it in a convenient format for data analysis.

Navigating through the MLBAM website can be overwhelming, but it helps to recognize that a homepage exists for nearly every day and every game. For example, http://gd2.mlb.com/components/game/mlb/year_2011/month_02/day_26/ displays numerous hyperlinks to various files specific to February 26th, 2011. On this page is a hyperlink to `miniscoreboard.xml` which contains information on every game played on that date. This page also has numerous hyperlinks to game specific pages. For example, `gid_2011_02_26_phimlb_nyamlb_1/` points to the homepage for that day's game between the NY Yankees and Philadelphia Phillies. On this page is a hyperlink to the `players.xml` file which contains information about the players, umpires, and coaches (positions, names, batting average, etc.) coming into that game.

Starting from a particular game's homepage and clicking on the `inning/` directory, we *should* see another page with links to the `inning_all.xml` file and the `inning_hit.xml` file. If it is available, the `inning_all.xml` file contains the PITCHf/x data for that game. It's important to note that this file won't exist for some games, because some games are played in venues that do not have a working PITCHf/x system in place. This is especially

²Please be respectful of this service and store any information after you extract it instead of repeatedly querying the website. Before using any content from this website, please also read the copyright.

true for preseason games and games played prior to the 2008 season when the PITCHf/x system became widely adopted.³ The `inning_hit.xml` files have manually recorded spatial coordinates of where a home run landed or where the baseball made initial contact with a defender after it was hit into play.

The relationship between these XML files and the tables returned by `scrape` is presented in Table ?? . The `pitch` table is extracted from files whose name ends in `inning_all.xml`. This is the only table returned by `scrape` that contains data on the pitch-by-pitch level. The `atbat`, `runner`, `action` and `hip` tables from this same file are commonly labeled somewhat ambiguously as play-by-play data. The `player`, `coach`, and `umpire` tables are extracted from `players.xml` and are classified as game-by-game since there is one record per person per game. Figure 2.1 shows how these tables can be connected to one another in a database setting. The direction of the arrows represent a one to possibly many relationship. For example, at least one pitch is thrown for each *at bat* (that is, each bout between pitcher and batter) and there are numerous at bats within each game.

In a rough sense, one can relate tables returned by `scrape` back to XML nodes in the XML files. For convenience, some information in certain XML nodes are combined into one table. For example, information gleaned from the ‘top’, ‘bottom’, and ‘inning’ XML nodes within `inning_all.xml` are included as `inning` and `inning_side` fields in the `pitch`, `po`, `atbat`, `runner`, and `action` tables. This helps reduce the burden of merging many tables together in order to have inning information on the play-by-play and/or pitch-by-pitch level. Other information is simply ignored simply because it is redundant. For example, the ‘game’ node within the `players.xml` file contains information that can be recovered from the `game` table extracted from the `miniscoreboard.xml` file. If the reader wants a more detailed explanation of fields in these tables, Marchi and Albert (2013) provide nice overview.

³In this case, `scrape` will print “failed to load HTTP resource” in the R console (after the relevant file name) to indicate that no data was available.

Table 2.1 Structure of PITCHf/x and related Gameday data sources accessible to ‘scrape()’

| Source file | Information | XML nodes | Tables Returned |
|--------------------|---------------------------------|--|--|
| miniscoreboard.xml | game-by-game | games, game game_media, media | game, media |
| players.xml | game-by-game | game, team, player, coach, umpire | player, coach, umpire |
| inning_all.xml | play-by-play, pitch-by-pitch | game, inning, bottom, top, atbat, po, pitch, runner action | atbat, po, pitch, runner, action |
| inning_hit.xml | play-by-play | hitchart, hip | hip |

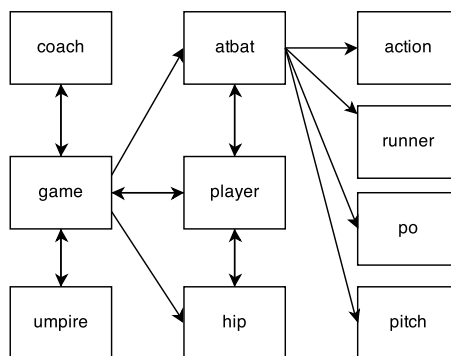


Figure 2.1 Table relations between Gameday data accessible via ‘scrape()’. The direction of the arrows indicate a one to possibly many relationship.

2.3 Introducing XML2R

XML2R adds to the CRAN Task View on Web Technologies and Services by focusing on the transformation of XML content into a collection of tables. Compared to a lower-level API like the **XML** package, it can significantly reduce the amount of coding and cognitive effort required to perform such a task. In contrast to most higher-level APIs, it does not make assumptions about the XML structure or its source. Although **XML2R** works on any structure, performance and user experience are enhanced if the content has an inherent relational structure. **XML2R**'s novel approach to XML data collection breaks down the transformation process into a few simple steps and allows the user to decide how to apply those steps.

The next few sections demonstrate how **pitchRx** leverages **XML2R** in order to produce a collection of tables from `inning_all.xml` files. A similar approach is used by `pitchRx::scrape` to construct tables from the other Gameday files in Table ?? . In fact, **XML2R** has also been used in the R package `bb scrapeR` which collects data from `nba.com` and `wnba.com`.

2.3.1 Constructing file names

Sometimes the most frustrating part of obtaining data in bulk off of the web is finding the proper collection of URLs or file names of interest. Since files on the Gameday website are fairly well organized, the `makeUrls` function can be used to construct `urls` that point to every game's homepage within a window of dates.

```
urls <- pitchRx::makeUrls(start = "2011-06-01", end = "2011-06-01")
sub("http://gd2.mlb.com/components/game/mlb/", "", head(urls))
#> [1] "year_2011/month_06/day_01/gid_2011_06_01_anamlb_kcamlb_1"
```

```
#> [2] "year_2011/month_06/day_01/gid_2011_06_01_balmlb_seamlb_1"
#> [3] "year_2011/month_06/day_01/gid_2011_06_01_chamlb_bosmlb_1"
#> [4] "year_2011/month_06/day_01/gid_2011_06_01_clemlb_tormlb_1"
#> [5] "year_2011/month_06/day_01/gid_2011_06_01_colmlb_lanmlb_1"
#> [6] "year_2011/month_06/day_01/gid_2011_06_01_flomlb_arimlb_1"
```

2.3.2 Extracting observations

Once we have a collection of XML files, the next step is to parse the content into a list of *observations*. An observation is technically defined as a matrix with one row and some number of columns. The columns are defined by XML attributes and the XML value (if any) for a particular XML lineage. The name of each observation tracks the XML hierarchy so observations can be grouped together in a sensible fashion at a later point.

```
library(XML2R)
files <- paste0(urls, "/inning/inning_all.xml")
obs <- XML2Obs(files, url.map = TRUE, quiet = TRUE)
table(names(obs))
```

| | | |
|----|-------------------------------------|---------------------------------|
| #> | | |
| #> | game | game//inning |
| #> | 15 | 137 |
| #> | game//inning//bottom//action | game//inning//bottom//atbat |
| #> | 116 | 532 |
| #> | game//inning//bottom//atbat//pitch | game//inning//bottom//atbat//po |
| #> | 1978 | 61 |
| #> | game//inning//bottom//atbat//runner | game//inning//top//action |
| #> | 373 | 150 |

```

#>           game//inning//top//atbat      game//inning//top//atbat//pitch
#>                               593                               2183
#>           game//inning//top//atbat//po      game//inning//top//atbat//runner
#>                               75                               509
#>                               url_map
#>                               1

```

This output tells us that 247 pitches were thrown in the bottom inning and 278 were thrown in the top inning on June 1st, 2011. Also, there are 12 different levels of observations. The list element named `url_map` is not considered an observation and was included since `url.map = TRUE`. This helps avoid repeating long file names in the `url_key` column which tracks the mapping between observations and file names.

```

obs[1]
#> $`game//inning//top//atbat//pitch`
#>      des      des_es      id  type tfs
#> [1,] "Called Strike" "Strike cantado" "3" "S" "201107"
#>      tfs_zulu      x      y      event_num sv_id
#> [1,] "2011-06-01T20:11:07Z" "103.00" "149.38" "3"      "110601_151108"
#>      play_guid start_speed end_speed sz_top sz_bot pfx_x  pfx_z  px
#> [1,] ""          "94.0"      "86.1"      "2.85" "1.36" "-8.12" "11.0" "-0.143"
#>      pz      x0      y0      z0      vx0      vy0      vz0      ax
#> [1,] "2.376" "-2.435" "50.0" "5.831" "9.058" "-137.334" "-7.288" "-15.446"
#>      ay      az      break_y break_angle break_length pitch_type
#> [1,] "31.474" "-11.175" "23.8"  "46.3"      "4.0"      "FF"
#>      type_confidence zone nasty spin_dir  spin_rate  cc mt url_key
#> [1,] ".865"          "2"  "46"  "216.336" "2753.789" "" "" "url1"

```


2.3.3 Renaming observations

Before grouping observations into a collection tables based on their names, one may want to `re_name` observations. Observations with names `'game//inning//bottom//atbat'` and `'game//inning//top//atbat'` should be included in the same table since they share XML attributes (in other words, the observations share variables).

```
tmp <- re_name(obs, equiv = c("game//inning//top//atbat",
                              "game//inning//bottom//atbat"), diff.name = "inning_side")
```

By passing these names to the `equiv` argument, `re_name` determines the difference in the naming scheme and suppresses that difference. In other words, observation names that match the `equiv` argument will be renamed to `'game//inning//atbat'`. The information removed from the name is not lost; however, as a new column is appended to the end of each relevant observation. For example, notice how the `inning_side` column contains the part of the name we just removed:

```
tmp[grepl("game//inning//atbat", names(tmp))][1:2]
#> $`game//inning//atbat`
#>      num b    s    o    start_tfs start_tfs_zulu      batter    stand
#> [1,] "1" "3" "2" "0" "201034" "2011-06-01T20:10:34Z" "430947" "L"
#>      b_height pitcher p_throws
#> [1,] "5-10" "462956" "R"
#>      des
#> [1,] "Erick Aybar singles on a line drive to center fielder Melky Cabrera. "
#>      des_es
#> [1,] "Erick Aybar pega sencillo con línea a jardinero central Melky Cabrera. "
#>      event_num event      event_es    home_team_runs away_team_runs url_key
#> [1,] "12"      "Single" "Sencillo" "0"      "0"      "url1"
```

```

#>      inning_side
#> [1,] "top"
#>
#> $`game//inning//atbat`
#>      num b    s    o    start_tfs start_tfs_zulu      batter    stand
#> [1,] "2" "2" "3" "1" "201412" "2011-06-01T20:14:12Z" "110029" "L"
#>      b_height pitcher p_throws des
#> [1,] "6-0"      "462956" "R"      "Bobby Abreu called out on strikes.  "
#>      des_es                                     event_num event      event_es
#> [1,] "Bobby Abreu se poncha sin tirarle.  " "24"      "Strikeout" "Ponche"
#>      home_team_runs away_team_runs url_key inning_side
#> [1,] "0"              "0"              "url1"  "top"

```

For similar reasons, other observation name pairs are renamed in a similar fashion.

```

tmp <- re_name(tmp, equiv = c("game//inning//top//atbat//runner",
  "game//inning//bottom//atbat//runner"), diff.name = "inning_side")
tmp <- re_name(tmp, equiv = c("game//inning//top//action",
  "game//inning//bottom//action"), diff.name = "inning_side")
tmp <- re_name(tmp, equiv = c("game//inning//top//atbat//po",
  "game//inning//bottom//atbat//po"), diff.name = "inning_side")
obs2 <- re_name(tmp, equiv = c("game//inning//top//atbat//pitch",
  "game//inning//bottom//atbat//pitch"), diff.name = "inning_side")
table(names(obs2))
#>
#>
#>      game      game//inning
#>      15      137
#>      game//inning//action  game//inning//atbat

```

```

#>                                266                                1125
#>  game//inning//atbat//pitch    game//inning//atbat//po
#>                                4161                                136
#>  game//inning//atbat//runner                                url_map
#>                                882                                1

```

2.3.4 Linking observations

After all that renaming, we now have 7 different levels of observations. Let's examine the first three observations on the `game//inning` level:

```

obs2[grep("^game//inning$", names(obs2))][1:3]
#> $`game//inning`
#>      num away_team home_team next url_key
#> [1,]  "1"  "ana"      "kca"      "Y"  "url1"
#>
#> $`game//inning`
#>      num away_team home_team next url_key
#> [1,]  "2"  "ana"      "kca"      "Y"  "url1"
#>
#> $`game//inning`
#>      num away_team home_team next url_key
#> [1,]  "3"  "ana"      "kca"      "Y"  "url1"

```

Before grouping observations into tables, it is usually important preserve the parent-to-child relationships in the XML lineage. For example, one may want to map a particular pitch back to the inning in which it was thrown. Using the `add_key` function, the relevant value of `num` for `game//inning` observations can be `recycled` to its XML descendants.

```
obskey <- add_key(obs2, parent = "game//inning", recycle = "num", key.name = "inning")
```

As it turns out, the `away_team` and `home_team` columns are redundant as this information is embedded in the `url` column. Thus, there is only one other informative attribute on this level which is `next`. By recycling this value among its descendants, we remove any need to retain a `game//inning` table.

```
obskey <- add_key(obskey, parent = "game//inning", recycle = "next")
```

It is also imperative that we can link a `pitch`, `runner`, or `po` back to a particular `atbat`. This can be done as follows:

```
obskey <- add_key(obskey, parent = "game//inning//atbat", recycle = "num")
```

2.3.5 Collapsing observations

Finally, we are in a position to pool together observations that have a common name. The `collapse_obs` function achieves this by row binding observations with the same name together and returning a list of matrices. Note that `collapse_obs` does not require that observations from the same level to have the same set of variables in order to be bound into a common table. In the case where variables are missing, `NA`s will be inserted as values.

```
tables <- collapse_obs(obskey)
#As mentioned before, we do not need the 'inning' table
tables <- tables[!grepl("^game//inning$", names(tables))]
table.names <- c("game", "action", "atbat", "pitch", "po", "runner")
tables <- setNames(tables, table.names)
head(tables[["runner"]])
```

```

#>      id      start end  event      event_num url_key
#> [1,] "430947" ""    "1B" "Single"      "12"      "url1"
#> [2,] "430947" "1B"  "2B" "Stolen Base 2B"  "19"      "url1"
#> [3,] "430947" "2B"  "3B" "Groundout"    "30"      "url1"
#> [4,] "430947" "3B"   ""  "Groundout"    "36"      "url1"
#> [5,] "543333" ""    "1B" "Single"      "58"      "url1"
#> [6,] "543333" "1B"   ""  "Pickoff Attempt 1B" "69"      "url1"

#>      inning_side inning next num score rbi earned
#> [1,] "top"        "1"    "Y"  "1" NA     NA     NA
#> [2,] "top"        "1"    "Y"  "2" NA     NA     NA
#> [3,] "top"        "1"    "Y"  "3" NA     NA     NA
#> [4,] "top"        "1"    "Y"  "4" NA     NA     NA
#> [5,] "bottom"     "1"    "Y"  "7" NA     NA     NA
#> [6,] "bottom"     "1"    "Y"  "8" NA     NA     NA

```

2.4 Collecting Gameday data with pitchRx

The main scraping function in **pitchRx**, **scrape**, can be used to easily obtain data from the files listed in Table ?? . In fact, any combination of these files can be queried using the **suffix** argument. In the example below, the **start** and **end** arguments are also used so that all available file types for June 1st, 2011 are queried.

```

library(pitchRx)

files <- c("inning/inning_all.xml", "inning/inning_hit.xml",
          "miniscoreboard.xml", "players.xml")

dat <- scrape(start = "2011-06-01", end = "2011-06-01", suffix = files)

```

The `game.ids` option can be used instead of `start` and `end` to obtain an equivalent `dat` object. This option can be useful if the user wants to query specific games rather than all games played over a particular time span. When using this `game.ids` option, the built-in `gids` object, is quite convenient.

```
data(gids, package = "pitchRx")
gids11 <- gids[grep("2011_06_01", gids)]
head(gids11)

#> [1] "gid_2011_06_01_anamlb_kcamlb_1" "gid_2011_06_01_balmlb_seamlb_1"
#> [3] "gid_2011_06_01_chamlb_bosmlb_1" "gid_2011_06_01_clemlb_tormlb_1"
#> [5] "gid_2011_06_01_colmlb_lanmlb_1" "gid_2011_06_01_flomlb_arimlb_1"

dat <- scrape(game.ids = gids11, suffix = files)
```

The object `dat` is a list of data frames containing all data available for June 1st, 2011 using `scrape`. The list names match the table names provided in Table ???. For example, `dat$atbat` is data frame with every at bat on June 1st, 2011 and `dat$pitch` has information related to the outcome of each pitch (including PITCHf/x parameters). The `object.size` of `dat` is nearly 300MB. Multiplying this number by 100 days exceeds the memory of most machines. Thus, if a large amount of data is required, the user should exploit the R database interface (R Special Interest Group on Databases 2013).

2.5 Storing and querying Gameday data

Since PITCHf/x data can easily exhaust memory, one should consider establishing a database instance before using `scrape`. By passing a database connection to the `connect` argument, `scrape` will try to create (and/or append to existing) tables using that connection. If the connection fails for some reason, tables will be written as csv files

in the current working directory. The benefits of using the `connect` argument includes improved memory management which can greatly reduce run time. `connect` will support a MySQL connection, but creating a SQLite database is quite easy with **dplyr** (Wickham and Francois 2014).

```
library(dplyr)

db <- src_sqlite("GamedayDB.sqlite3", create = TRUE)

# Collect and store all PITCHf/x data from 2008 to now

scrape(start = "2008-01-01", end = Sys.Date(),
        suffix = "inning/inning_all.xml", connect = db$con)
```

In the later sections, animations of four-seam and cut fastballs thrown by Mariano Rivera and Phil Hughes during the 2011 season are created. In order to obtain the data for those animations, one could query `db` which now has PITCHf/x data from 2008 to date. This query requires criteria on: the `pitcher_name` field (in the `atbat` table), the `pitch_type` field (in the `pitch` table), and the `date` field (in both tables). To reduce the time required to search those records, one should create an index on each of these three fields.

```
library(DBI)

dbSendQuery(db$con, "CREATE INDEX pitcher_index ON atbat(pitcher_name)")
dbSendQuery(db$con, "CREATE INDEX type_index ON pitch(pitch_type)")
dbSendQuery(db$con, "CREATE INDEX date_atbat ON atbat(date)")
```

As a part of our query, we'll have to join the `atbat` table together with the `pitch` table. For this task, the `gameday_link` and `num` fields are helpful since together they provide a way to match pitches with at bats. For this reason, a multi-column index on the `gameday_link` and `num` fields will further reduce run time of the query.

```
dbSendQuery(db$con, 'CREATE INDEX pitch_join ON pitch(gameday_link, num)')
dbSendQuery(db$con, 'CREATE INDEX atbat_join ON atbat(gameday_link, num)')
```

Although the query itself could be expressed entirely in SQL, **dplyr**'s grammar for data manipulation (which is database agnostic) can help to simplify the task. In this case, `at.bat` is a tabular *representation* of the remote `atbat` table restricted to cases where Rivera or Hughes was the pitcher. That is, `at.bat` does not contain the actual data, but it does contain the information necessary to retrieve it from the database.

```
at.bat <- tbl(db, "atbat") %>%
  filter(pitcher_name %in% c("Mariano Rivera", "Phil Hughes"))
```

Similarly, `fbs` is a tabular representation of the `pitch` table restricted to four-seam (FF) and cut fastballs (FC).

```
fbs <- tbl(db, "pitch") %>%
  filter(pitch_type %in% c("FF", "FC"))
```

An `inner_join` of these two filtered tables returns a tabular representation of all four-seam and cut fastballs thrown by Rivera and Hughes. Before `collect` actually performs the database query and brings the relevant data into the R session, another restriction is added so that only pitches from 2011 are included.

```
pitches <- inner_join(fbs, at.bat) %>%
  filter(date >= "2011_01_01" & date <= "2012_01_01") %>%
  collect()
```


2.6 Visualizing PITCHf/x

2.6.1 Strike-zone plots and umpire bias

Amongst the most common PITCHf/x graphics are strike-zone plots. Such a plot has two axes and the coordinates represent the location of baseballs as they cross home plate. The term strike-zone plot can refer to either *density* or *probabilistic* plots. Density plots are useful for exploring what *actually* occurred, but probabilistic plots can help address much more interesting questions using statistical inference. Although probabilistic plots can be used to visually track any event probability across the strike-zone, their most popular use is for addressing umpire bias in a strike versus ball decision (Green and Daniels 2014). The probabilistic plots section demonstrates how **pitchRx** simplifies the process behind creating such plots via a case study on the impact of home field advantage on umpire decisions.

In the world of sports, it is a common belief that umpires (or referees) have a tendency to favor the home team. PITCHf/x provides a unique opportunity to add to this discussion by modeling the probability of a called strike at home games versus away games. Specifically, conditioned upon the umpire making a decision at a specific location in the strike-zone, if the probability that a home pitcher receives a called strike is higher than the probability that an away pitcher receives a called strike, then there is evidence to support umpire bias towards a home pitcher.

There are many different possible outcomes of each pitch, but we can condition on the umpire making a decision by limiting to the following two cases. A *called strike* is an outcome of a pitch where the batter does not swing and the umpire declares the pitch a strike (which is a favorable outcome for the pitcher). A *ball* is another outcome where the batter does not swing and the umpire declares the pitch a ball (which is a favorable

outcome for the batter). All decisions made between 2008 and 2013 can be obtained from `db` with the following query using **dplyr**.

```
# First, add an index on the pitch description to speed up run-time
dbSendQuery(db$con, "CREATE INDEX des_index ON pitch(des)")

pitch <- tbl(db, "pitch") %>%
  filter(des %in% c("Called Strike", "Ball")) %>%
  # Keep pitch location, descriptions
  select(px, pz, des, gameday_link, num) %>%
  # 0-1 indicator of strike/ball
  mutate(strike = as.numeric(des == "Called Strike"))

atbat <- tbl(db, "atbat") %>%
  # Select variables to be used later as covariates in probabilistic models
  select(b_height, p_throws, stand, inning_side, date, gameday_link, num)

decisions <- inner_join(pitch, atbat) %>%
  filter(date <= "2014_01_01") %>%
  collect()
```

2.6.1.1 Density plots

The `decisions` data frame contains data on over 2.5 million pitches thrown from 2008 to 2013. About a third of them are called strikes and two-thirds balls. Figure 2.2 shows the density of all called strikes. Clearly, most called strikes occur on the outer region of the strike-zone. Many factors could contribute to this phenomenon; which we will not investigate here.

```

# strikeFX uses the stand variable to calculate strike-zones
# Here is a slick way to create better facet titles without changing data values
relabel <- function(variable, value) {
  value <- sub("^R$", "Right-Handed Batter", value)
  sub("^L$", "Left-Handed Batter", value)
}
strikes <- subset(decisions, strike == 1)
strikeFX(strikes, geom = "raster", layer = facet_grid(. ~ stand, labeller = relabel))

```

Figure 2.2 shows one static rectangle (or strike-zone) per plot automatically generated by `strikeFX`. The definition of the strike-zone is notoriously ambiguous. As a result, the boundaries of the strike-zone may be noticeably different in some situations. However, we can achieve a fairly accurate representation of strike-zones using a rectangle defined by batters' average height and stance (Fast 2011). As Figure 2.4 reinforces, batter stance makes an important difference since the strike-zone seems to be horizontally shifted away from the batter. The batter's height is also important since the strike-zone is classically defined as approximately between the batter's knees and armpits.

Figure 2.2 has is one strike-zone per plot since the `layer` option contains a `ggplot2` argument that facets according to batter stance. Facet layers are a powerful tool for analyzing PITCHf/x data because they help produce quick and insightful comparisons. In addition to using the `layer` option, one can add layers to a graphic returned by `strikeFX` using `ggplot2` arithmetic. It is also worth pointing out that Figure 2.2 could have been created without introducing the `strikes` data frame by using the `density1` and `density2` options.

```

strikeFX(decisions, geom = "raster", density1 = list(des = "Called Strike"),
  density2 = list(des = "Called Strike")) + facet_grid(. ~ stand, labeller = relabel)

```

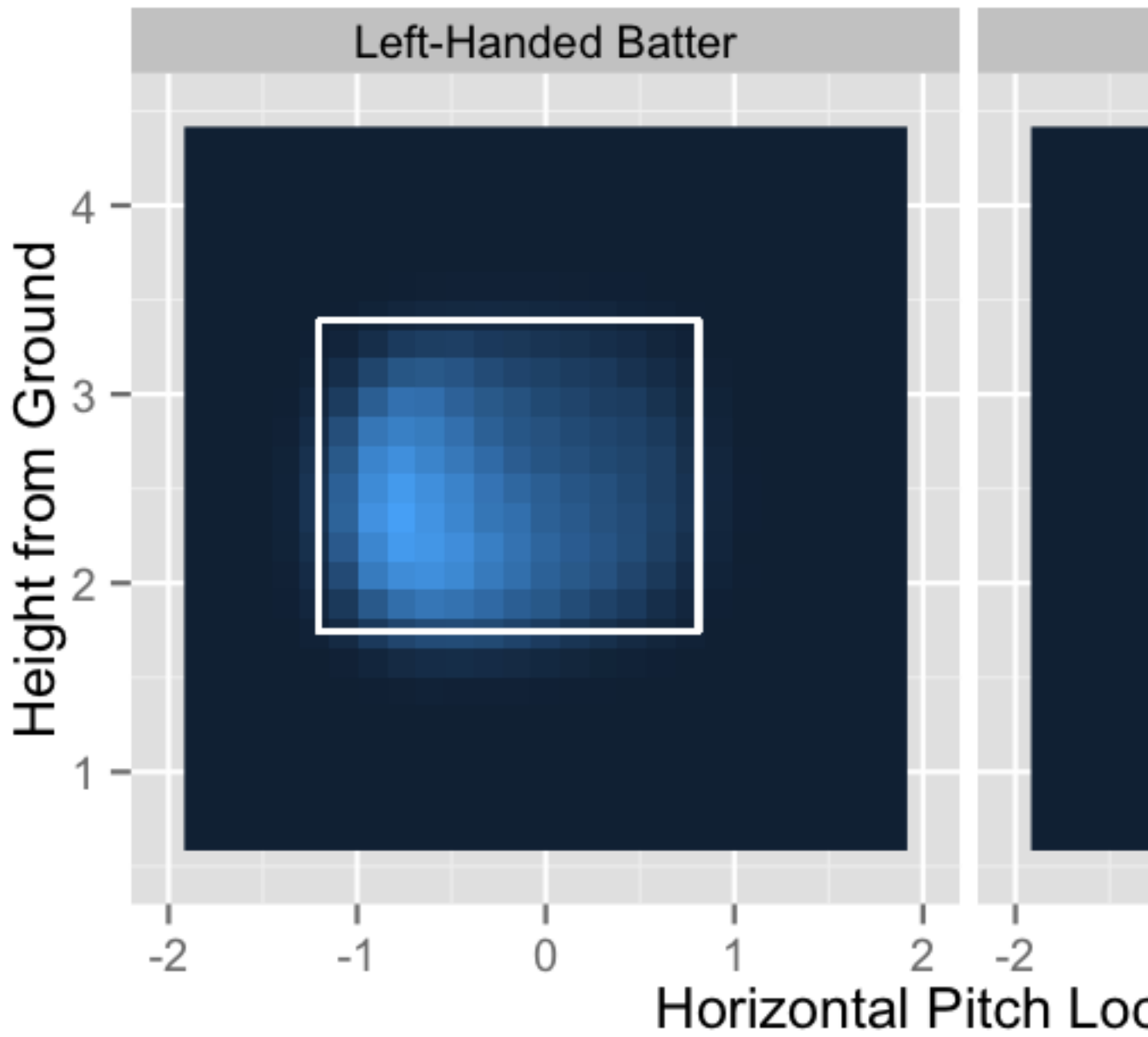


Figure 2.2 Density of called strikes for right-handed batters and left-handed batters (from 2008 to 2013).

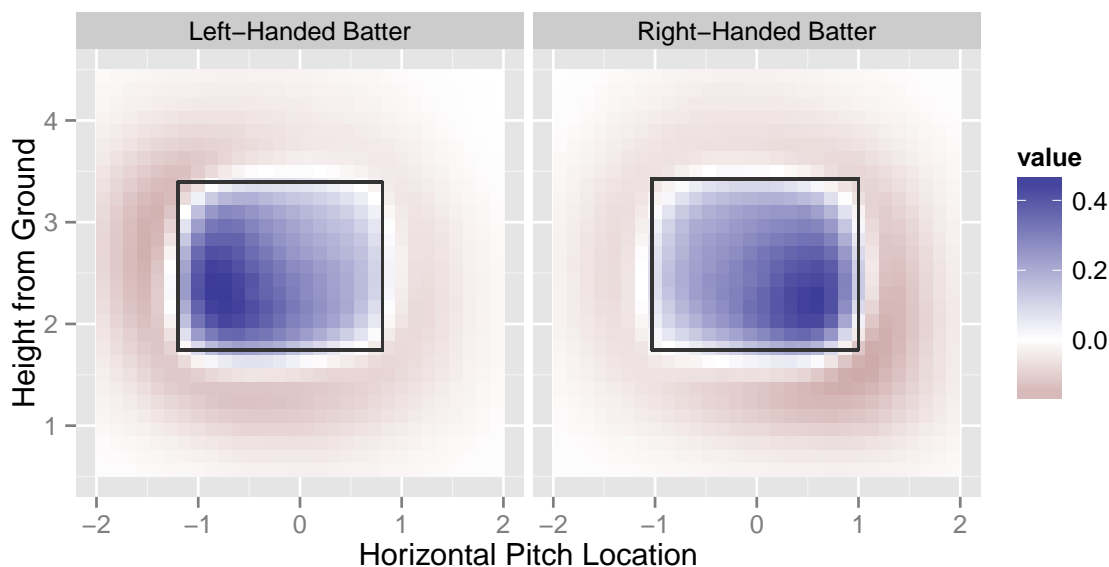


Figure 2.3 Density of called strikes minus density of balls for both right-handed batters and left-handed batters (from 2008 to 2013). The blue region indicates a higher frequency of called strikes and the red region indicates a higher frequency of balls.

In general, when `density1` and `density2` are identical, the result is equivalent to subsetting the data frame appropriately beforehand. More importantly, by specifying *different* values for `density1` and `density2`, differenced densities are easily generated. In this case, a grid of density estimates for `density2` are subtracted from the corresponding grid of density estimates for `density1`. Note that the default `NULL` value for either density option infers that the entire data set defines the relevant density. Thus, if `density2` was `NULL` (when `density1 = list(des = 'Called Strike')`), we would obtain the density of called strikes minus the density of *both* called strikes and balls. In Figure 2.3, we define `density1` as called strikes and define `density2` as balls. As expected, we see positive density values (in blue) inside the strike-zone and negative density values (in red) outside of the strike-zone.

```
strikeFX(decisions, geom = "raster", density1 = list(des = "Called Strike"),
  density2 = list(des = "Ball"), layer = facet_grid(. ~ stand, labeller = relabel))
```

These density plots are helpful for visualizing the observed frequency of events; however, they are not very useful for addressing our umpire bias hypothesis. Instead of looking simply at the *density*, we want to model the *probability* of a strike called at each coordinate given the umpire has to make a decision.

2.6.1.2 Probabilistic plots

There are many approaches to probabilistic modeling over a two dimensional spatial region. Since our response is often categorical, generalized additive models (GAMs) is a popular and desirable approach to modeling events over the strike-zone (Mills 2010). There are numerous R package implementations of GAMs, but the `bam` function from the `mgcv` package has several desirable properties (Wood 2006). Most importantly, the smoothing parameter can be estimated using several different methods. In order to have a reasonable estimate of the smooth 2D surface, GAMs require fairly large amount of observations. As a result, run time can be an issue – especially when modeling 2.5 million observations! Thankfully, the `bam` function has a `cluster` argument which allows one to distribute computations across multiple cores using the built in `parallel` package.

```
library(parallel)
cl <- makeCluster(detectCores() - 1)
library(mgcv)
m <- bam(strike ~ interaction(stand, p_throws, inning_side) +
  s(px, pz, by = interaction(stand, p_throws, inning_side)),
  data = decisions, family = binomial(link = 'logit'), cluster = cl)
```

This formula models the probability of a strike as a function of the baseball’s spatial location, the batter’s stance, the pitcher’s throwing arm, and the side of the inning. Since home pitchers always pitch during the top of the inning, `inning_side` also serves as an

indication of whether a pitch is thrown by a home pitcher. In this case, the `interaction` function creates a factor with eight different levels since every input factor has two levels. Consequently, there are 8 different levels of smooth surfaces over the spatial region defined by `px` and `pz`.

The fitted model `m` contains a lot of information which `strikeFX` uses in conjunction with any `ggplot2` facet commands to infer which and how surfaces should be plotted. In particular, the `var.summary` is used to identify model covariates, as well their default conditioning values. In our case, the majority of `decisions` are from right-handed pitchers and the top of the inning. Thus, the default conditioning values are `"top"` for `inning_side` and `"R"` for `p_throws`. If different conditioning values are desired, `var.summary` can be modified accordingly. To demonstrate, Figure 2.4 shows 2 of the 8 possible surfaces that correspond to a right-handed *away* pitcher.

```
away <- list(inning_side = factor("bottom", levels = c("top", "bottom")))
m$var.summary <- modifyList(m$var.summary, away)
strikeFX(decisions, model = m, layer = facet_grid(. ~ stand, labeller = relabel))
```

Using the same intuition exploited earlier to obtain differenced density plots, we can easily obtain differenced probability plots. To obtain Figure 2.5, we simply add `p_throws` as another facet variable and `inning_side` as a differencing variable. In this case, conditioning values do not matter since every one of the 8 surfaces are required in order to produce Figure 2.5.

```
# Function to create better labels for both stand and p_throws
relabel2 <- function(variable, value) {
  if (variable %in% "stand")
    return(sub("^L$", "Left-Handed Batter",
              sub("^R$", "Right-Handed Batter", value)))
}
```

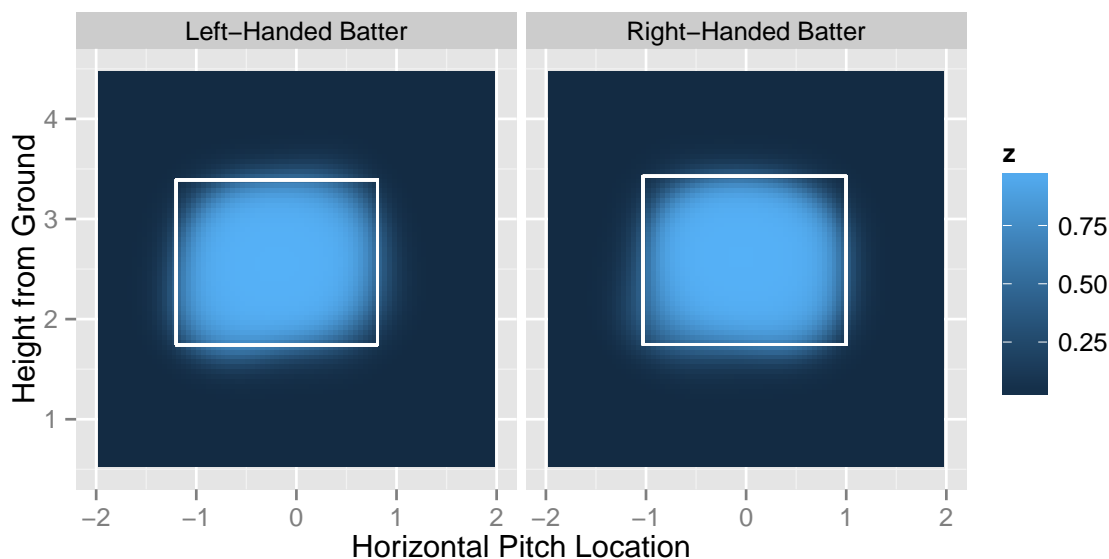


Figure 2.4 Probability that a right-handed away pitcher receives a called strike (provided the umpire has to make a decision). Plots are faceted by the handedness of the batter.

```
if (variable %in% "p_throws")
  return(sub("^L$", "Left-Handed Pitcher",
    sub("^R$", "Right-Handed Pitcher", value)))
}

strikeFX(decisions, model = m, layer = facet_grid(p_throws ~ stand, labeller = relabeller),
  density1 = list(inning_side = "top"), density2 = list(inning_side = "bottom"))
```

The four different plots in Figure 2.5 represent the four different combination of values among `p_throws` and `stand`. In general, provided that a pitcher throws to a batter in the blue region, the pitch is more likely to be called a strike if the pitcher is on their home turf. Interestingly, there is a well-defined blue elliptical band around the boundaries of the typical strike-zone. Thus, home pitchers are more likely to receive a favorable call – especially when the classification of the pitch is in question. In some areas, the home pitcher has up to a 6 percent higher probability of receiving a called strike than

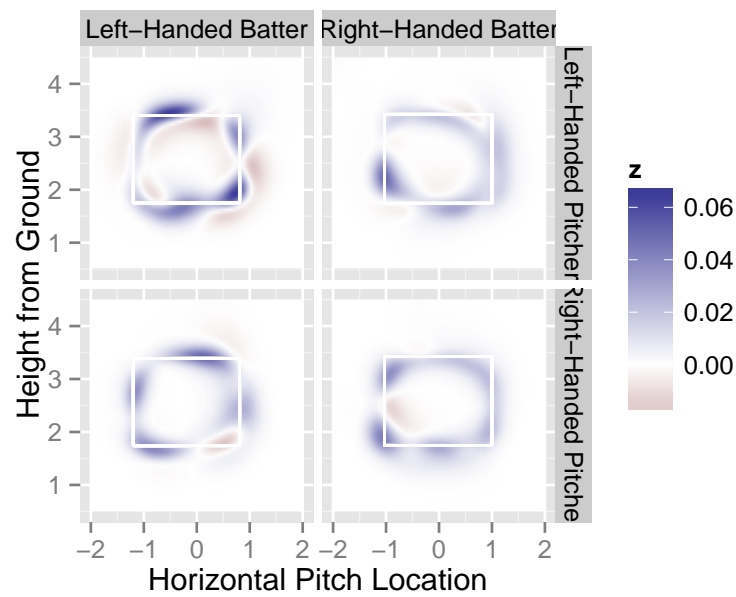


Figure 2.5 Difference between home and away pitchers in the probability of a strike (provided the umpire has to make a decision). The blue regions indicate a higher probability of a strike for home pitchers and red regions indicate a higher probability of a strike for away pitchers. Plots are faceted by the handedness of both the pitcher and the batter.

an away pitcher. The subtle differences in spatial patterns across the different values of `p_throws` and `stand` are interesting as well. For instance, pitching at home has a large positive impact for a left-handed pitcher throwing in the lower inside portion of the strike-zone to a right-handed batter, but the impact seems negligible in the mirror opposite case. Differenced probabilistic densities are clearly an interesting visual tool for analyzing PITCHf/x data. With `strikeFX`, one can quickly and easily make all sorts of visual comparisons for various situations. In fact, one can explore and compare the probabilistic structure of any well-defined event over a strike-zone region (for example, the probability a batter reaches base) using a similar approach.

2.6.2 2D animation

`animateFX` provides convenient and flexible functionality for animating the trajectory of any desired set of pitches. For demonstration purposes, this section animates every four-seam and cut fastball thrown by Mariano Rivera and Phil Hughes during the 2011 season. These pitches provide a good example of how facets play an important role in extracting new insights. Similar methods can be used to analyze any MLB player (or combination of players) in greater detail.

`animateFX` tracks three dimensional pitch locations over a sequence of two dimensional plots. The animation takes on the viewpoint of the umpire; that is, each time the plot refreshes, the balls are getting closer to the viewer. This is reflected with the increase in size of the points as the animation progresses. Obviously, some pitches travel faster than others, which explains the different sizes within a particular frame. Animations revert to the initial point of release once *all* of the baseballs have reached home plate. During an interactive session, `animateFX` produces a series of plots that may not be viewed easily. One

option available to the user is to wrap `animation::saveHTML` around `animateFX` to view the animation in a browser with proper looping controls (Xie 2013).

To reduce the time and thinking required to produce these animations, `animateFX` has default settings for the geometry, color, opacity and size associated with each plot. Any of these assumptions can be altered - except for the point geometry. In order for animations to work, a data frame with the appropriately named PITCHf/x parameters (that is, `x0`, `y0`, `z0`, `vx0`, `vy0`, `vz0`, `ax0`, `ay0` and `az0`) is required. In Figure 2.6, every four-seam and cut fastball thrown by Rivera and Hughes during the 2011 season is visualized using the `pitches` data frame obtained earlier (the animation is available at <http://cpsievert.github.io/pitchRx/ani1>).

```
animateFX(pitches, layer=list(theme_bw(), coord_equal(),
  facet_grid(pitcher_name~stand, labeller = relabel)))
```

In the animation corresponding to Figure 2.6, the upper right-hand portion (Rivera throwing to right-handed batters) reveals the clearest pattern in flight trajectories. Around the point of release, Rivera's two pitch types are hard to distinguish. However, after a certain point, there is a very different flight path among the two pitch types. Specifically, the drastic left-to-right movement of the cut fastball is noticeably different from the slight right-to-left movement of the four-seam fastball. In recent years, cut fastballs have gained notoriety among the baseball community as a coveted pitch for pitchers have at their disposal. This is largely due to the difficulty that a batter has in distinguishing the cut fastball from another fastball as the ball travels toward home plate. Clearly, this presents an advantage for the pitcher since they can use deception to reduce batter's ability to predict where the ball will cross home plate. This deception factor combined with Rivera's ability to locate his pitches explain his accolades as one of the greatest pitchers of all time (Traub 2010).

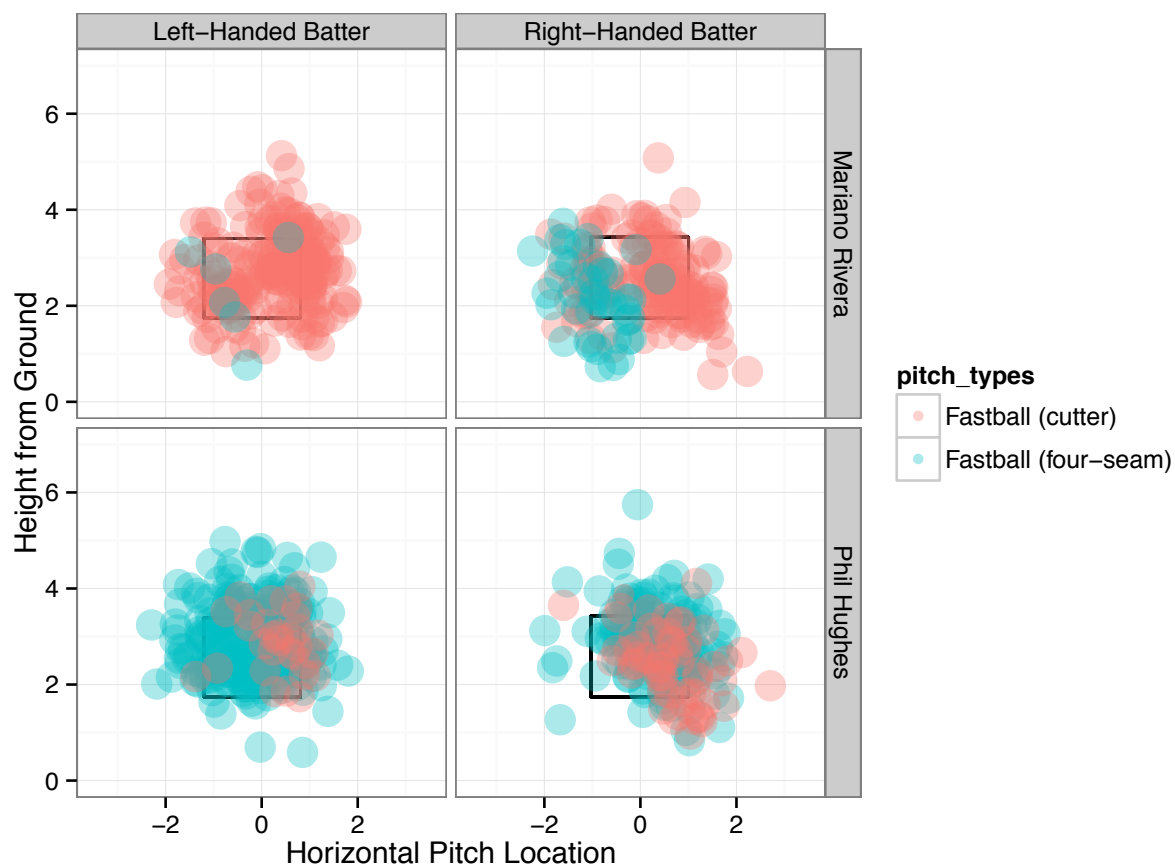


Figure 2.6 The last frame of an animation of every four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. The actual animation can be viewed at <http://cpsievert.github.io/pitchRx/ani1>. Pitches are faceted by pitcher and batting stance. For instance, the top left plot portrays pitches thrown by Rivera to left-handed batters.

Although we see a clear pattern in Rivera's pitches, MLB pitchers are hardly ever that predictable. Animating that many pitches for another pitcher can produce a very cluttered graphic which is hard to interpret (especially when many pitch types are considered). However, we may still want to obtain an indication of pitch trajectory over a set of many pitches. A way to achieve this is to average over the PITCHf/x parameters to produce an overall sense of pitch type behavior (via the `avg.by` option). Note that the facet variables are automatically considered indexing variables. That is, in Figure 2.7, there are eight 'average' pitches since there are two pitch types, two pitchers, and two types of batting stance (the animation is available at <http://cpsievert.github.io/pitchRx/ani2>).

```
animateFX(pitches, avg.by = "pitch_types", layer = list(coord_equal(), theme_bw(),
  facet_grid(pitcher_name~stand, labeller = relabel)))
```

2.6.3 Interactive 3D graphics

rgl is an R package that utilizes OpenGL for graphics rendering. **interactiveFX** utilizes **rgl** functionality to reproduce flight paths on an interactive 3D platform. Figure 2.8 has two static pictures of Mariano Rivera's 2011 fastballs on this interactive platform. This is great for gaining new perspectives on a certain set of pitches, since the trajectories can be viewed from any angle. Figure 2.8 showcases the difference in trajectory between Rivera's pitch types.

```
Rivera <- subset(pitches, pitcher_name == "Mariano Rivera")
interactiveFX(Rivera, avg.by = "pitch_types")
```

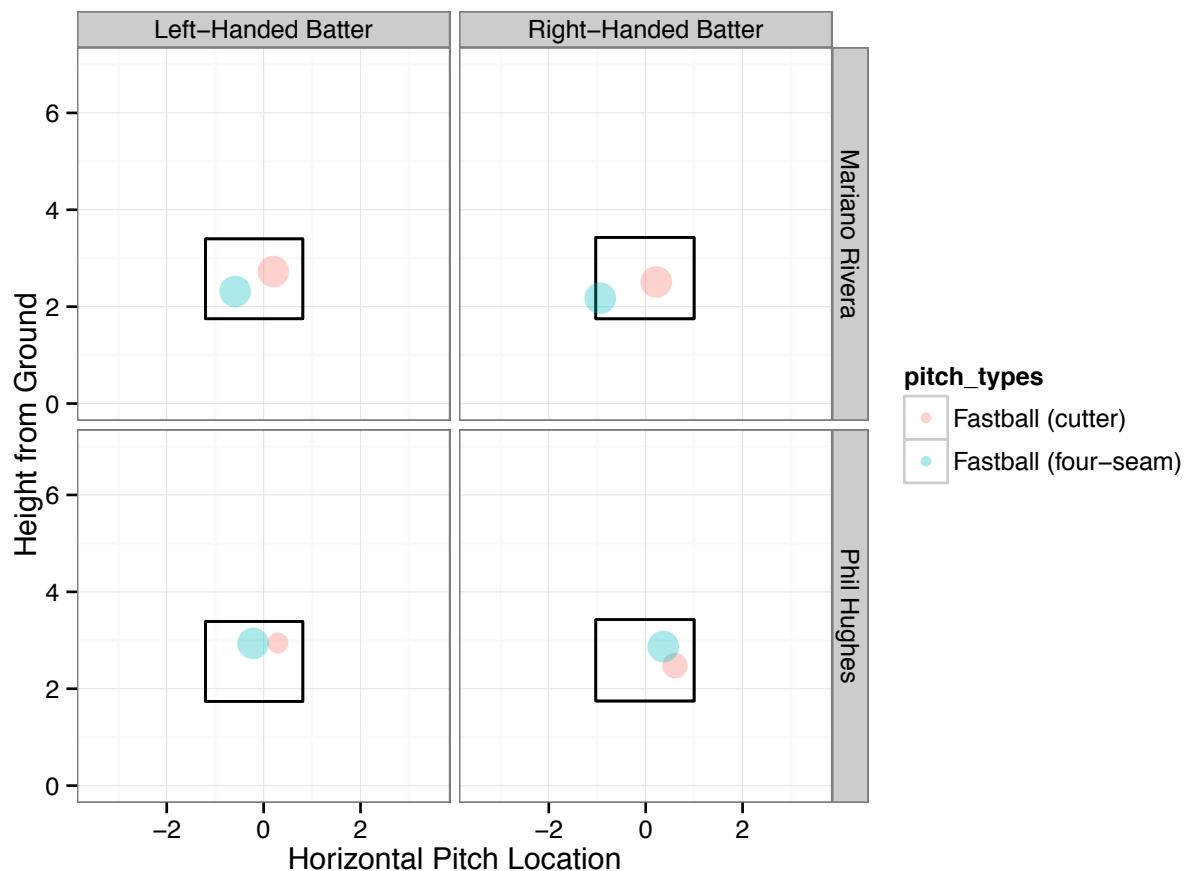


Figure 2.7 The last frame of an animation of averaged four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. The actual animation can be viewed at <http://cpsievert.github.io/pitchRx/ani2>. PITCHf/x parameters are averaged over pitch type, pitcher and batting stance. For instance, the bottom right plot portrays an average four-seam and average cutter thrown by Hughes to right-handed batters.

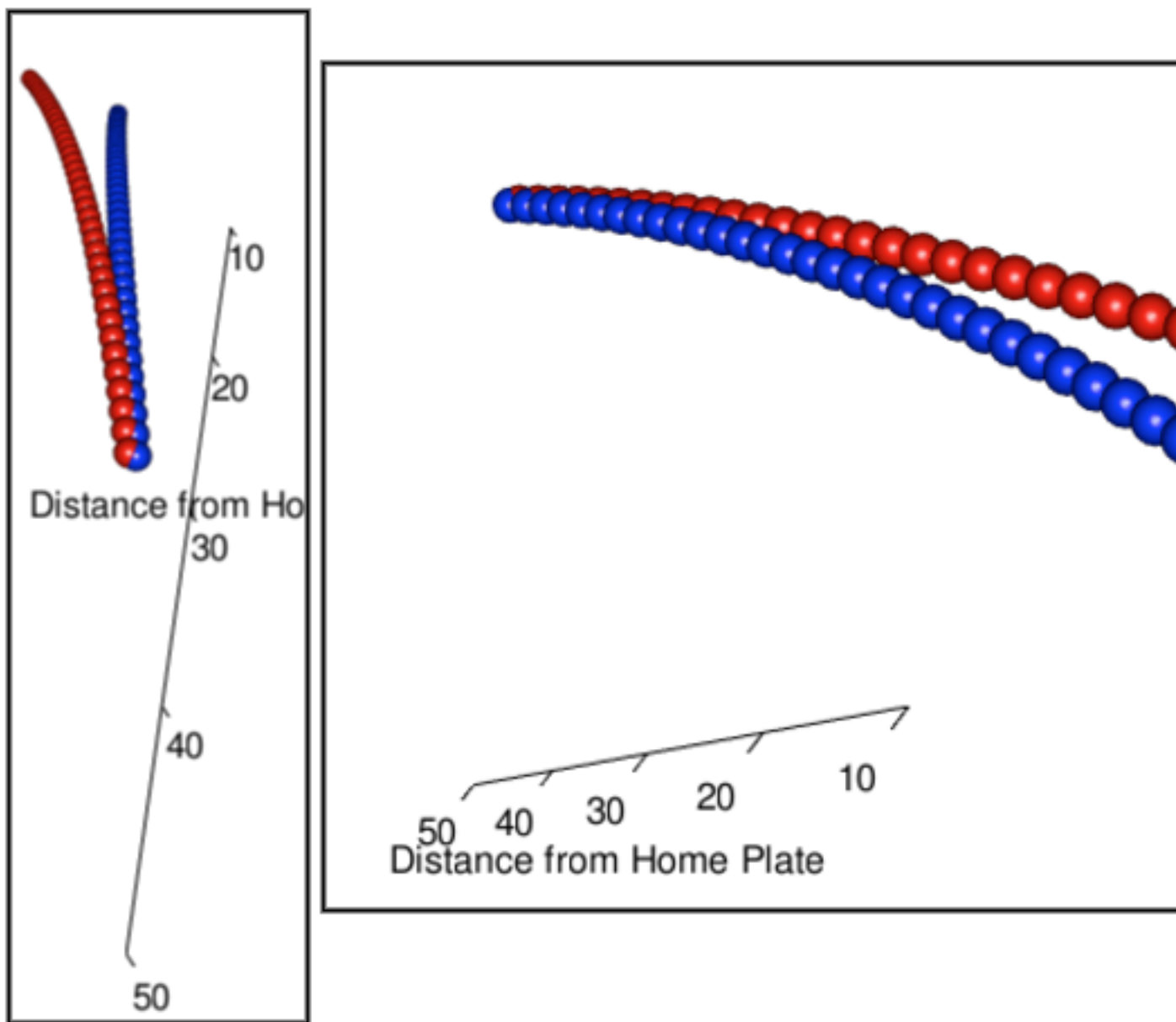


Figure 2.8 3D scatterplot of pitches from Rivera. Pitches are plotted every one-hundredth of a second. Cutting fastballs are shown in red and four-seam fastballs are shown in blue. The left hand plot takes a viewpoint of Rivera and the right hand plot takes a viewpoint near the umpire. Note these are static pictures of an interactive object.

2.7 Conclusion

pitchRx utilizes **XML2R**'s convenient framework for manipulating XML content in order to provide easy access to PITCHf/x and related Gameday data. **pitchRx** removes access barriers which allows the average R user and baseball fan to spend their valuable time analyzing Gameday's enormous source of baseball information. **pitchRx** also provides a suite of functions that greatly reduce the amount of work involved to create popular PITCHf/x graphics. For those interested in obtaining other XML data, **pitchRx** serves as a nice example of leveraging **XML2R** to quickly assemble custom XML data collection mechanisms.

3 LDAvis: A method for visualizing and interpreting topics

This chapter is a paper published in The Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces (ACL 2014) [Sievert:2014b]. I am the primary author of the paper which is available online here <http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf>

The formatting of paper has been modified to make for consistent typesetting across the thesis.

ABSTRACT

We present **LDAvis**, an **R** package for creating We present **LDAvis**, a web-based interactive visualization of topics estimated using Latent Dirichlet Allocation that is built using a combination of **R** and **d3**. Our visualization provides a global view of the topics (and how they differ from each other), while at the same time allowing for a deep inspection of the tokens most highly associated with each individual topic. First, we propose a novel method for choosing which tokens to present to a user to aid in the task of topic interpretation, in which we define the *relevance* of a token to a topic. Second, we present the results of a user study that illustrates how ranking tokens by their relevance to a given topic relates to that topic’s interpretability, and we recommend a default method of computing relevance to maximize topic interpretability. Last, we incorporate relevance into **LDAvis** in a way that allows users to flexibly explore topic-token relationships to better understand a fitted LDA model.

3.1 Introduction

Recently much attention has been paid to visualizing the output of topic models fit using Latent Dirichlet Allocation (LDA) (Matthew J. Gardner and Seppi 2010); (Chaney and Blei 2012); (Jason Chuang and Heer 2012b); (Brynjar Gretarsson and Smyth 2011). Such visualizations are challenging to create because of the high dimensionality of the fitted model – LDA is typically applied to thousands of documents, which are modeled as mixtures of dozens of topics, which themselves are modeled as distributions over

thousands of tokens (David M. Blei and Jordan 2012); (Griffiths and Steyvers 2004). The most promising basic technique for creating LDA visualizations that are both compact and thorough is *interactivity*.

We introduce an interactive visualization system that we call **LDavis** that attempts to answer a few basic questions about a fitted topic model: (1) What is the meaning of each topic?, (2) How prevalent is each topic?, and (3) How do the topics relate to each other? Different visual components answer each of these questions, some of which are original, and some of which are borrowed from existing tools.

Our visualization (illustrated in Figure 3.1) has two basic pieces. First, the left panel of our visualization presents a “global” view of the topic model, and answers questions 2 and 3. In this view, we plot the topics as circles in the two-dimensional plane whose centers are determined by computing the distance between topics (using a distance measure of the user’s choice) and then by using multidimensional scaling to project the inter-topic distances onto two dimensions, as is done in (Jason Chuang and Heer 2012a). We encode each topic’s overall prevalence using the areas of the circles, where we sort the topics in decreasing order of prevalence.

Second, the right panel of our visualization depicts a horizontal barchart whose bars represent the individual tokens that are the most useful for interpreting the currently selected topic on the left, and allows users to answer question 1, “What is the meaning of each topic?”. A pair of overlaid bars represent both the corpus-wide frequency of a given token as well as the topic-specific frequency of the token, as in (Jason Chuang and Heer 2012b).

The left and right panels of our visualization are linked such that selecting a topic (on the left) reveals the most useful tokens (on the right) for interpreting the selected topic. In addition, selecting a token (on the right) reveals the conditional distribution over topics

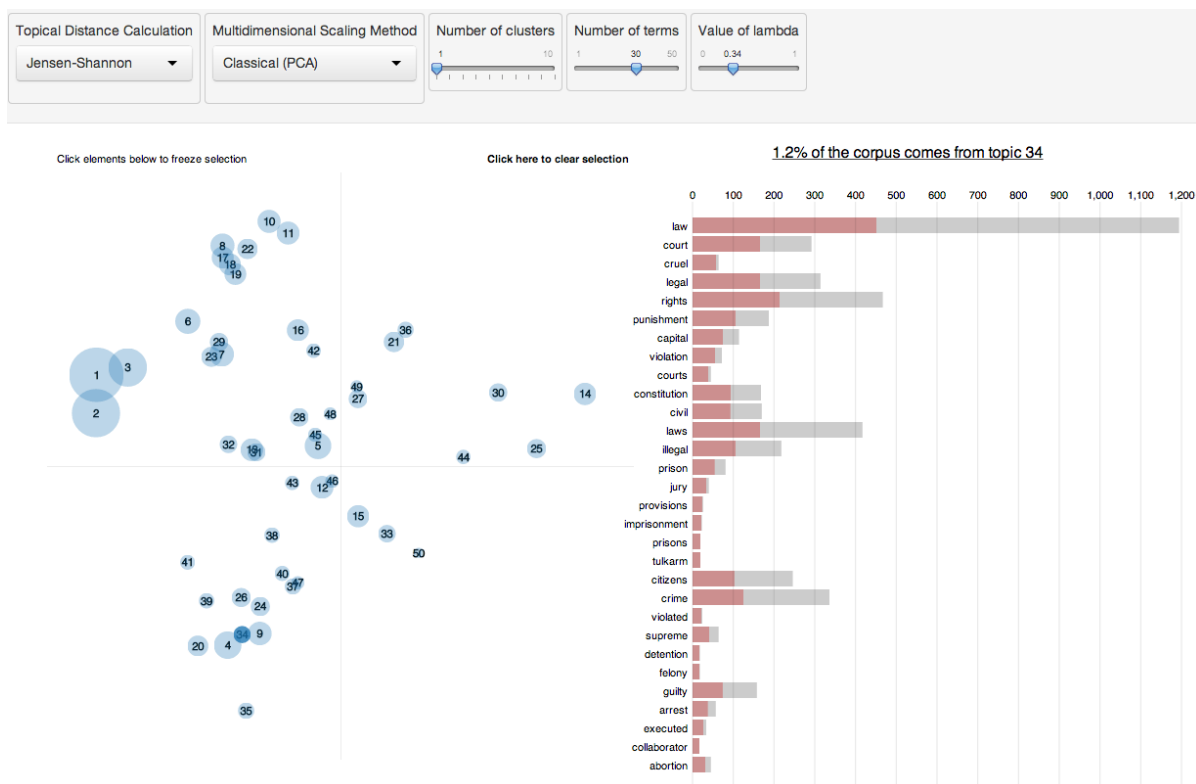


Figure 3.1 The layout of LDAvis, with the global topic view on the left, and the token barcharts on the right. Linked selections allow users to reveal aspects of the topic-token relationships compactly.

(on the left) for the selected token. This kind of linked selection allows users to examine a large number of topic-token relationships in a compact manner.

A key innovation of our system is how we determine the most useful tokens for interpreting a given topic, and how we allow users to interactively adjust this determination. A topic in LDA is a multinomial distribution over the tokens in the vocabulary, where the vocabulary typically contains thousands of tokens. To interpret a topic, one typically examines a ranked list of the most probable tokens in that topic, using anywhere from three to thirty tokens in the list. The problem with interpreting topics this way is that common tokens in the corpus often appear near the top of such lists for multiple topics, making it hard to differentiate the meanings of these topics.

Bischof and Airoldi (2012) propose ranking tokens for a given topic in terms of both the *frequency* of the token under that topic as well as the token’s *exclusivity* to the topic, which accounts for the degree to which it appears in that particular topic to the exclusion of others. We propose a similar measure that we call the *relevance* of a token to a topic to create a flexible method for ranking tokens in order of usefulness for interpreting topics. We discuss our definition of relevance, and its graphical interpretation, in detail in Section 3.3.1. We also present the results of a user study conducted to determine the optimal tuning parameter in the definition of relevance to aid the task of topic interpretation in Section 3.3.2, and we describe how we incorporate relevance into our interactive visualization in Section 3.4.

3.2 Related Work

Much work has been done recently regarding the interpretation of topics (i.e. measuring topic “coherence”) as well as visualization of topic models.

3.2.1 Topic Interpretation and Coherence

It is well-known that the topics inferred by LDA are not always easily interpretable by humans. Jonathan Chang and Blei (2009) established via a large user study that standard quantitative measures of fit, such as those summarized by Hanna M. Wallach and Mimno (2009), do not necessarily agree with measures of topic interpretability by humans. Daniel Ramage et al. (2009) assert that “characterizing topics is hard” and describe how using the top- k tokens for a given topic might not always be best, but offer few concrete alternatives.

Loulwah AlSumait and Domeniconi (2009), David Mimno and McCallum (2011), and Jason Chuang and Heer (2013b) develop quantitative methods for measuring the interpretability of topics based on experiments with data sets that come with some notion of topical ground truth, such as document metadata or expert-created topic labels. These methods are useful for understanding, in a global sense, which topics are interpretable (and why), but they don’t specifically attempt to aid the user in interpreting *individual* topics.

Blei and Lafferty (2009) developed “Turbo Topics”, a method of identifying n-grams within LDA-inferred topics that, when listed in decreasing order of probability, provide users with extra information about the usage of tokens within topics. This two-stage process yields good results on experimental data, although the resulting output is still simply a ranked list containing a mixture of tokens and n-grams, and the usefulness of the method for topic interpretation was not tested in a user study.

David Newman and Baldwin (2010) describe a method for ranking tokens within topics to aid interpretability called Pointwise Mutual Information (PMI) ranking. Under PMI ranking of tokens, each of the ten most probable tokens within a topic are ranked in decreasing order of approximately how often they occur in close proximity to the nine other most probable tokens from that topic in some large, external “reference” corpus, such as Wikipedia or Google n-grams. Although this method correlated highly with human judgments of token importance within topics, it does not easily generalize to topic models fit to corpora that don’t have a readily available external source of word co-occurrences.

In contrast, Taddy (2011) uses an intrinsic measure to rank tokens within topics: a quantity called *lift*, defined as the ratio of a token’s probability within a topic to its marginal probability across the corpus. This generally decreases the rankings of globally

frequent tokens, which can be helpful. We find that it can be noisy, however, by giving high rankings to very rare tokens that occur in only a single topic, for instance. While such tokens may contain useful topical content, if they are very rare the topic may remain difficult to interpret.

Finally, Bischof and Airoldi (2012) propose and implement a new statistical topic model that infers both a token’s frequency as well as its *exclusivity* – the degree to which its occurrences are limited to only a few topics. They introduce a univariate measure called a FREX score (“**FR**equency and **EX**clusivity”) which is a weighted harmonic mean of a token’s rank within a given topic with respect to frequency and exclusivity, and they recommend it as a way to rank tokens to aid topic interpretation. We propose a similar method that is a weighted average of a token’s probability and its lift, and we justify it with a user study and incorporate it into our interactive visualization.

3.2.2 Topic Model Visualization Systems

A number of visualization systems for topic models have arisen in recent years. Several of them focus on allowing users to browse documents, topics, and tokens to learn about the relationships between these three canonical topic model units (Matthew J. Gardner and Seppi 2010); (Chaney and Blei 2012) (Justin Snyder and Wolfe 2013). These browsers typically use lists of the most probable tokens within topics to summarize the topics, and the visualization elements are limited to barcharts or word clouds of token probabilities for each topic, pie charts of topic probabilities for each document, and/or various barcharts or scatterplots related to document metadata. Although these tools can be useful for browsing a corpus, we seek a more compact visualization, with the more narrow focus of quickly and easily understanding the individual topics themselves (without necessarily visualizing documents).

Jason Chuang and Heer (2012b) develop such a tool, called “Termite”, which visualizes the set of topic-token distributions estimated in LDA using a matrix layout. The authors introduce two measures of the usefulness of tokens for understanding a topic model: *distinctiveness* and *saliency*. These quantities measure how much information a token conveys about a topic by computing the Kullback-Liebler divergence between the distribution of topics given the token and the marginal distribution of topics (distinctiveness), optionally weighted by the token’s overall frequency (saliency). The authors recommend saliency as a thresholding method for selecting which tokens are included in the visualization, and they further use a seriation method for ordering the most salient tokens to highlight differences between topics.

Termite is a compact, intuitive interactive visualization of the topics in a topic model, but by only including tokens that rank high in saliency or distinctiveness, which are *global* properties of tokens, it is restricted to providing a *global* view of the model, rather than allowing a user to deeply inspect individual topics by visualizing a potentially different set of tokens for every single topic. In fact, Jason Chuang and Heer (2013a) describe the use of a “topic-specific word ordering” as potentially useful future work.

3.3 Relevance of tokens to topics

Here we define *relevance*, our method for ranking tokens within topics, and we describe the results of a user study to learn an optimal tuning parameter in the computation of relevance.

3.3.1 Definition of Relevance

Let ϕ_{kw} denote the probability of token $w \in \{1, \dots, V\}$ for topic $k \in \{1, \dots, K\}$, where V denotes the number of unique tokens in the vocabulary, and let p_w denote the marginal probability of token w in the corpus. One typically estimates ϕ in LDA using Variational Bayes methods or Collapsed Gibbs Sampling, and p_w from the empirical distribution of the corpus (optionally smoothed by including prior weights as pseudo-counts).

We define the *relevance* of token w to topic k given a weight parameter λ (where $0 \leq \lambda \leq 1$) as:

$$r(w, k \mid \lambda) = \lambda \log(\phi_{kw}) + (1 - \lambda) \log\left(\frac{\phi_{kw}}{p_w}\right),$$

where λ determines the weight given to the probability of token w under topic k relative to its lift. Setting $\lambda = 1$ results in the familiar ranking of tokens in decreasing order of their topic-specific probability, and setting $\lambda = 0$ ranks tokens solely by their lift, which we found anecdotally to result in “noisy” topics full of rare tokens. We wish to learn an “optimal” value of λ for topic interpretation from our user study.

First, though, to see how different values of λ result in different ranked token lists, consider the plot in Figure 3.2. We fit a 50-topic model to the 20 Newsgroups data (details are described in Section~3.3.2) and plotted $\log(\text{lift})$ on the y -axis vs. $\log(\phi_{kw})$ on the x -axis for each token in the vocabulary (which has size $V = 22,524$) for a given topic. Figure 3.2 shows this plot for Topic 29, which occurred mostly in documents posted to the “Motorcycles” newsgroup, but also from documents posted to the “Automobiles” newsgroup and the “Electronics” newsgroup. Graphically, the line separating the most relevant tokens for this topic, given λ , has slope $-\lambda/(1 - \lambda)$ (see Figure 3.2).

For this topic, the top-5 most relevant tokens given $\lambda = 1$ (ranking solely by probability) are {out, #emailaddress, #twodigitnumer, up, #onedigitnumber}, where a ‘#’ symbol

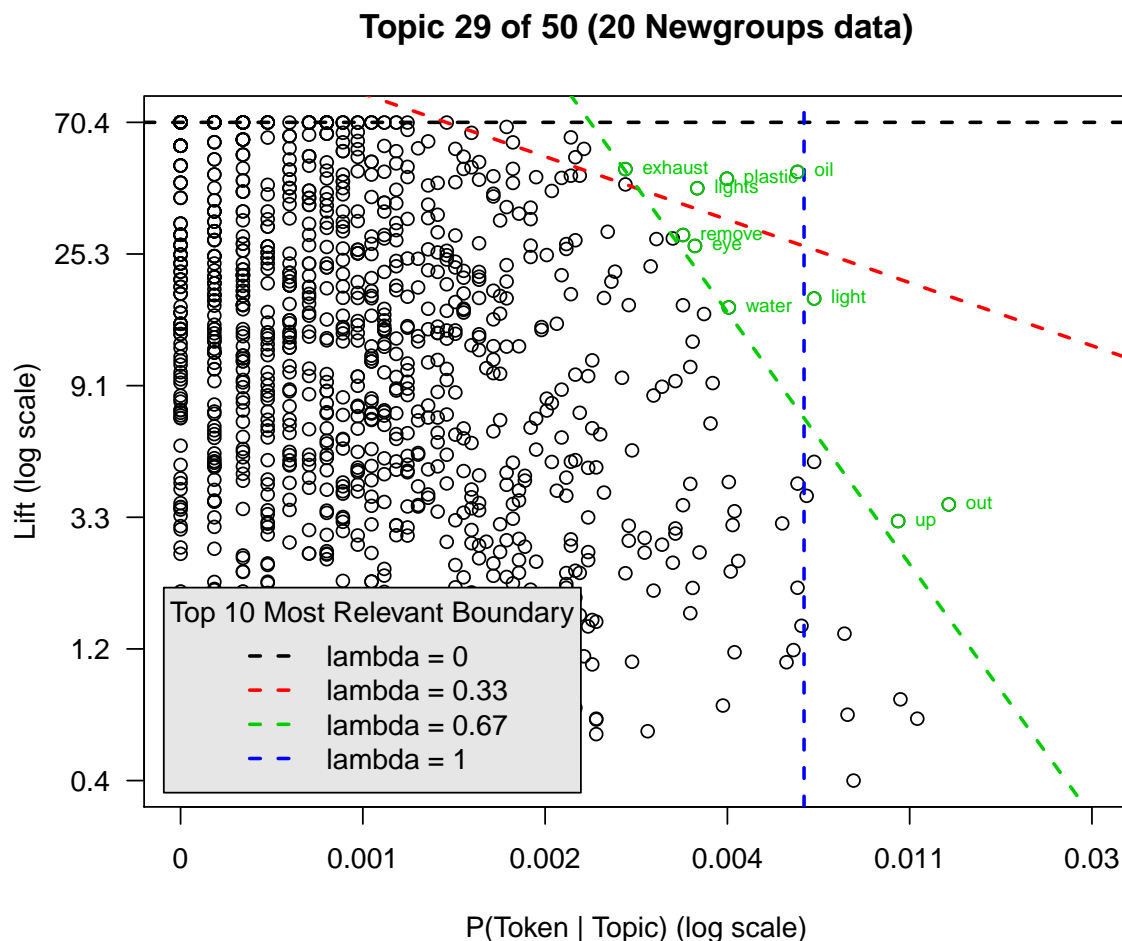


Figure 3.2 Dotted lines separating the top-10 most relevant tokens for different values of λ , with the most relevant tokens for $\lambda = 2/3$ displayed and highlighted in green.

denotes a token that is an entity representing a class of things. In contrast to this list, which contains globally common tokens and which provides very little meaning regarding motorcycles, automobiles, or electronics, the top-5 most relevant tokens given $\lambda = 1/3$ are {oil, plastic, pipes, fluid, and lights}. The second set of tokens is much more descriptive of the topic being discussed than the first.

3.3.2 User Study

We conducted a user study to determine whether there was an optimal value of λ in the definition of relevance to aid topic interpretation. First, we fit a 50-topic model to the $D = 13,695$ documents in the 20 Newsgroups data which were posted to a single Newsgroup (rather than two or more Newsgroups). We used the Collapsed Gibbs Sampler algorithm (Griffiths and Steyvers 2004) to sample the latent topics for each of the $N = 1,590,376$ tokens in the data, and we saved their topic assignments from the last iteration (after convergence). We then computed the 20 by 50 table, T , which contains, in cell T_{gk} , the count of the number of times a token from topic $k \in \{1, \dots, 50\}$ was assigned to Newsgroup $g \in \{1, \dots, 20\}$, where we defined the Newsgroup of a token to be the Newsgroup to which the document containing that token was posted. Some of the LDA-inferred topics occurred almost exclusively ($> 90\%$ of occurrences) in documents from a single Newsgroup, such as Topic 38, which was the estimated topic for 15,705 tokens in the corpus, 14,233 of which came from documents posted to the “Medicine” (or “sci.med”) Newsgroup. Other topics occurred in a wide variety of Newsgroups. One would expect these “spread-out” topics to be harder to interpret than the “pure” topics like Topic 38.

In the study we recruited 29 subjects among our colleagues, and each subject completed an online experiment consisting of 50 tasks, one for each topic in the fitted LDA model. Task k (for $k \in \{1, \dots, 50\}$) was to read a list of five tokens, ranked from 1-5 in terms of relevance to topic k , where $\lambda \in (0, 1)$ was randomly sampled to compute relevance. The user was instructed to identify which “topic” the list of tokens discussed from a list of three possible “topics”, where their choices were names of the Newsgroups. The correct answer for task k (i.e. our “ground truth”) was defined as the Newsgroup that contributed the most tokens to topic k (i.e. the Newsgroup with the largest count in the

k th column of the table T), and the two alternative choices were the Newsgroups that contributed the second and third-most tokens to topic k .

We anticipated that the effect of λ on the probability of a user making the correct choice could be different across topics. In particular, for “spread-out” topics that were inherently difficult to interpret, because their tokens were drawn from a wide variety of Newsgroups (similar to a “fused” topic in Jason Chuang and Heer (2013b)), we expected the proportion of correct responses to be roughly 1/3 no matter the value of λ used to compute relevance. Similarly, for very “pure” topics, whose tokens were drawn almost exclusively from one Newsgroup, we expected the task to be easy for any value of λ . To account for this, we analyzed the experimental data by fitting a varying-intercepts logistic regression model to allow each of the fifty topics to have its own baseline difficulty level, where the effect of λ is shared across topics. We used a quadratic function of λ in the model (linear, cubic and quartic functions were explored and rejected).

As expected, the baseline difficulty of each topic varied widely. In fact, seven of the topics were correctly identified by all 29 users, and one topic was incorrectly identified by all 29 users. For the remaining 42 topics we estimated a topic-specific intercept term to control for the inherent difficulty of identifying the topic (not just due to its tokens being spread among multiple Newsgroups, but also to account for the inherent familiarity of each topic to our subject pool – subjects, on average, were more familiar with “Cars” than “The X Window System”, for example).

The estimated effects of λ and λ^2 were 2.74 and -2.34, with standard errors 1.03 and 1.00. Taken together, their joint effect was statistically significant (χ^2 p-value = 0.018). %, but the signs of their coefficients agreed with our intuition, and in a similarly designed large-scale user study (on Mechanical Turk, for instance), we expect that their joint effect would be statistically significant. To see the estimated effect of λ on the probability of

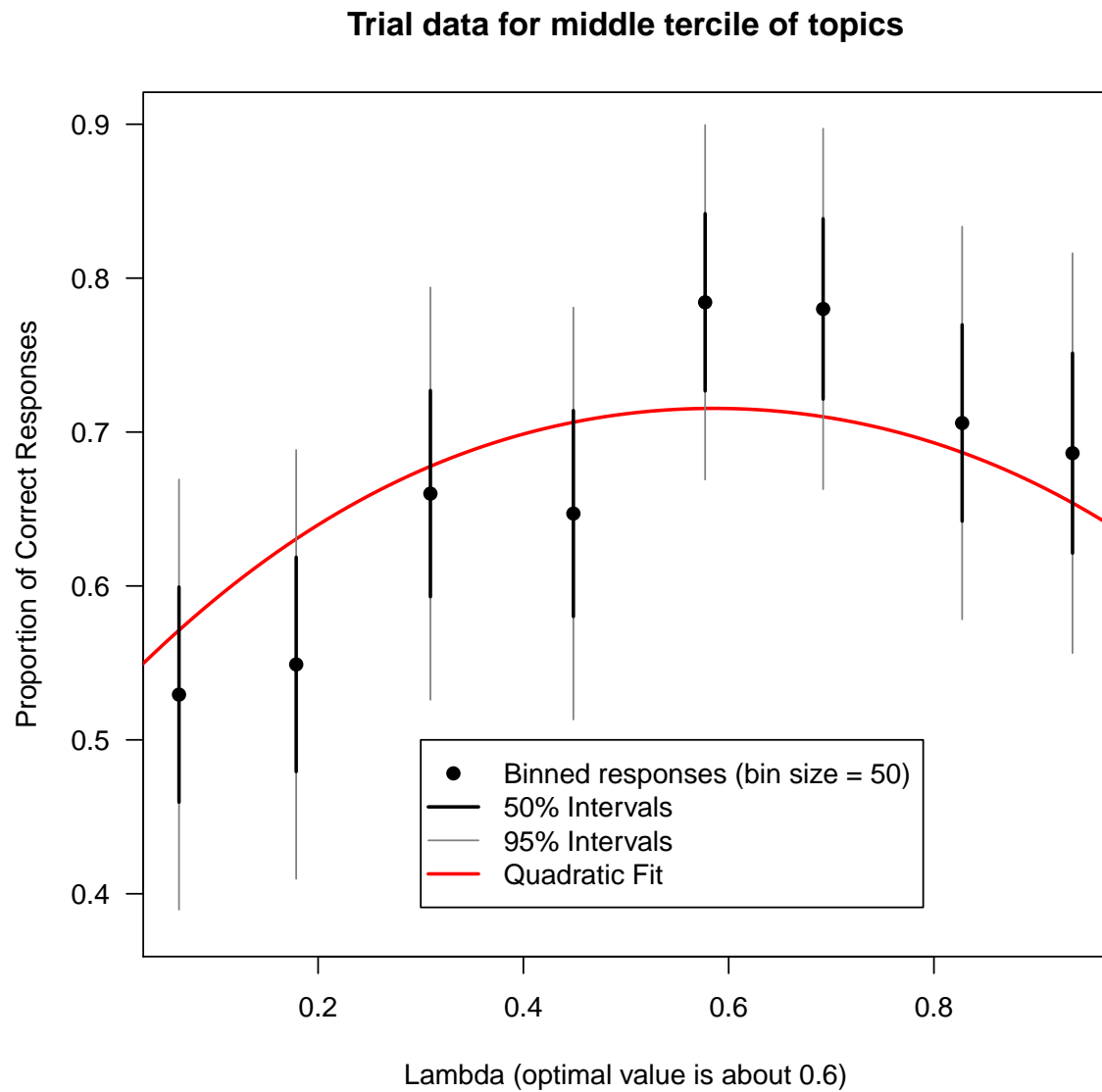


Figure 3.3 A plot of the proportion of correct responses in a user study vs. the value of λ used to compute the most relevant tokens for each topic.

correctly identifying a topic, consider Figure 3.3. We plot binned proportions of correct responses (on the y-axis) vs. λ (on the x-axis) for the 14 topics whose estimated topic-specific intercepts fell into the middle tercile among the 42 topics that weren’t trivial or impossible to identify. Among these topics there was roughly a 67% baseline probability of correct identification. As Figure 3.3 shows, for these topics, the “optimal” value of λ was about 0.6, and it resulted in a 70% - 75% probability of correct identification, whereas for values of λ near 0 or 1, the proportion of correct responses was closer to 55% or 60%. We view this as evidence that ranking tokens according to relevance, where $\lambda < 1$, can aid topic interpretation, even if this precise task (selecting a known topic label from a list of pre-defined labels associated with each document as metadata) is not always the goal. A similar conclusion might be drawn from an experiment to study the FREX token ranking method of Bischof and Airolidi (2012).

Note that in our experiment, we used the collection of single-posted 20 Newsgroups documents to define our “ground truth” data. An alternative method for collecting “ground truth” data would have been to recruit experts to label topics from an LDA model. We chose against this option because doing so would present a classic “chicken-or-egg” problem: If we use expert-labeled topics in an experiment to learn how to summarize topics so that they can be interpreted (i.e. “labeled”), we would only re-learn the way that our experts were instructed, or allowed, to label the topics in the first place! If, for instance, the experts were presented with a ranked list of the most probable tokens for each topic, this would influence the interpretations and labels they give to the topics, and the experimental result would be the circular conclusion that ranking tokens by probability allows users to recover the “expert” labels most easily. To avoid this, we felt strongly that we should use data in which documents have metadata associated with them. The 20 Newsgroups data provides an externally validated source of topic labels, in the sense that the labels were presented to users (in the form of Newsgroup names),

and users subsequently filled in the content. It represents, essentially, a crowd-sourced collection of tokens, or content, for a certain set of topic labels.

3.4 Our Visualization System

Our interactive, web-based visualization system, **LDavis**, has two core functionalities that enable users to understand the topic-token relationships in a fitted LDA model, and a number of extra features that provide additional perspectives on the model. %Usually these questions can not be answered easily with a few simple plots and/or metrics. Instead, an interactive layout such as **LDavis** allows one to quickly explore model output, form new hypotheses and verify findings.

First and foremost, **LDavis** allows one to select a topic to reveal the most relevant tokens for that topic. In Figure 3.1, Topic 34 is selected, and its 30 most relevant tokens (given $\lambda = 0.34$, in this case) populate the bar chart to the right (ranked in order of relevance from top to bottom). The widths of the gray bars represent the corpus-wide frequencies of each token, and the widths of the red bars represent the topic-specific frequencies of each token. A slider allows users to change the value of λ , which can alter the rankings of tokens to aid topic interpretation. By default, λ is set to 0.6, as suggested by our user study in Section~3.3.2. If $\lambda = 1$, tokens are ranked solely by ϕ_{kw} , which implies the red bars would be sorted from widest (at the top) to narrowest (at the bottom). By comparing the widths of the red and gray bars for a given token, users can quickly understand whether a token is highly relevant to the selected topic because of its lift (a high ratio of red to gray), or its probability (absolute width of red). The top 3 most relevant tokens in Figure 3.1 are “law”, “rights”, and “court”. Note that “law” is a common word which is generated by Topic 34 in about 40% of its corpus-wide occurrences, whereas “cruel” is a

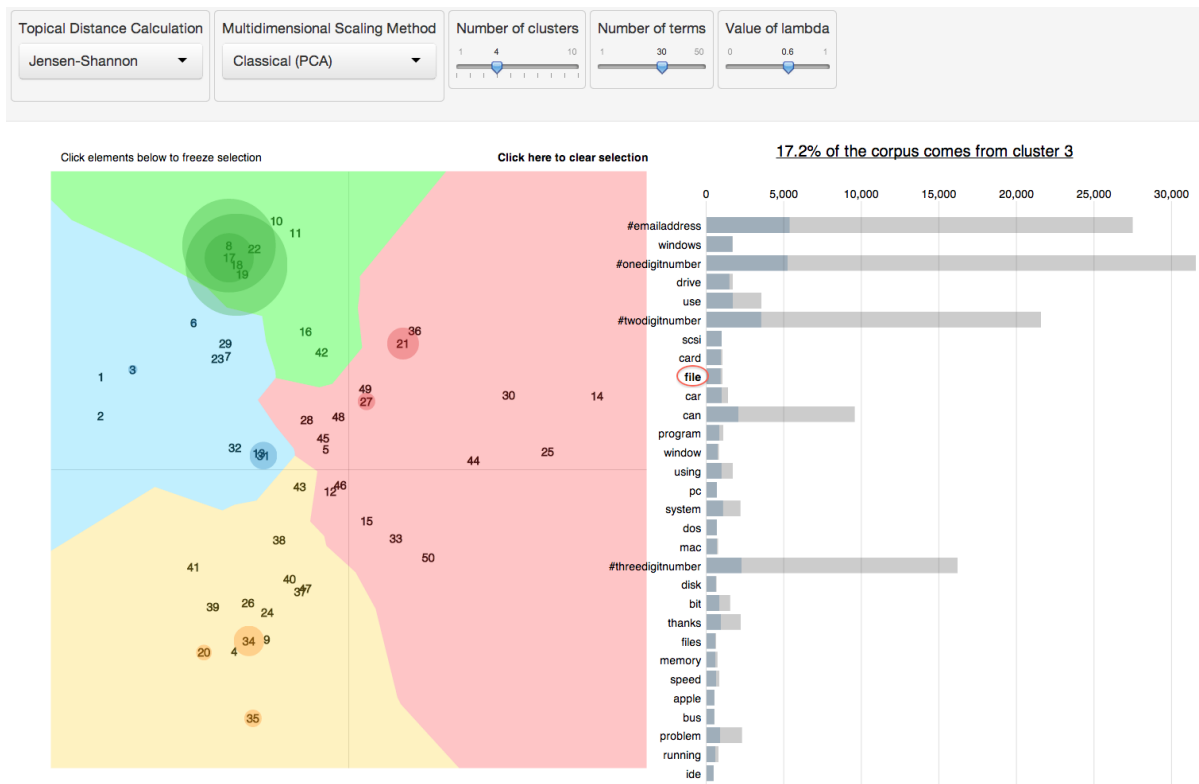


Figure 3.4 The user has chosen to segment the topics into four clusters, and has selected the green cluster to populate the barchart with the most relevant tokens for that cluster. Then, the user hovered over the ninth bar from the top, 'file', to display the conditional distribution over topics for this token.

relatively rare word with very high lift – it occurs almost exclusively in Topic 34. Such properties of the topic-token relationship are readily visible in **LDAvis** for every topic.

On the left panel, two visual features provide a global perspective of the topics. First, the areas of the circles are proportional to the relative prevalences of the topics in the corpus, θ_k , which can be computed as $\theta_k = \sum_d N_d \theta_{dk}$ for documents $d = 1, \dots, D$, where document d contains N_d tokens. In the 50-topic model fit to the 20 Newsgroups data, the first three topics comprise 12%, 9%, and 6% of the corpus, and all contain common, non-specific tokens (although there are differences: Topic 2 contains formal debate-related language such as “conclusion”, “evidence”, and “argument”, whereas Topic 3 contains slang conversational language such as “kinda”, “like”, and “yeah”). In addition to visualizing topic prevalence, the left pane shows inter-topic differences. The default for computing inter-topic distances is Jensen-Shannon divergence, although other metrics are enabled. The default for scaling the set of inter-topic distances defaults to Principal Components, but other other algorithms are also enabled.

The second core feature of **LDAvis** is the ability to select a token (by hovering over it) to reveal its conditional distribution over topics. This distribution is visualized by altering the areas of the topic circles such that they are proportional to the token-specific frequencies across the corpus. This allows the user to verify, as discussed in Jason Chuang and Heer (2012a), whether the multidimensional scaling of topics has faithfully clustered similar topics in two-dimensional space. For example, in Figure 3.4, the token “file” is selected. In the majority of this token’s occurrences, it is drawn from one of several topics located in the upper left-hand region of the global topic view. Upon inspection, this group of topics can be interpreted broadly as a discussion of computer hardware and software. This verifies, to some extent, their placement, via multidimensional scaling, into the same two-dimensional region. It also suggests that the word “file” used in this context refers to a computer file. However, there is also conditional probability mass for the token

“file” on Topic 34. As shown in Figure 3.1, Topic 34 can be interpreted as discussing the criminal punishment system where “file” refers to court filings. Similar discoveries can be made for any word that exhibits polysemy (such as “drive” appearing in computer- and automobile-related topics, or “ground” occurring in electrical- and baseball-related topics).

Beyond its within-browser interaction capability, **LDavis** leverages the **R** language to allow users to easily alter the topical distance measurement as well as the multidimensional scaling algorithm to produce the global topic view. In addition, there is an option to apply k -means clustering to the topics (as a function of their two-dimensional locations in the global topic view). This is merely an effort to facilitate semantic zooming in an LDA model with many topics where ‘after-the-fact’ clustering may be an easier way to learn clusters of topics, rather than fitting a hierarchical topic model (David M. Blei and Tenenbaum 2003), for example. Selecting a cluster (or region) of topics reveals the most relevant tokens for that group of topics, where the token distribution of a cluster of topics is defined as the average of the token distributions of the individual topics in the cluster. In Figure 3.4, the green cluster of topics is selected, and the most relevant tokens are predominantly related to computer hardware and software.

3.5 Discussion

We have described a web-based, interactive visualization system, **LDavis**, that enables deep inspection of topic-token relationships in an LDA model, while simultaneously providing a “global” view of the topics, via their prevalences and similarities to each other, in a compact space. We also propose a novel way to rank tokens within topics to aid in the task of topic interpretation, and we present a user study that attempts to not

only *measure* the interpretability of a topic, but also how to *maximize* the interpretability of the topic.

For future work, we anticipate performing a larger user study to further understand how to facilitate topic interpretation in fitted LDA models, including a comparison of multiple methods, such as ranking by Turbo Topics (Blei and Lafferty 2009) or FREX scores (Bischof and Airolidi 2012), in addition to relevance. We also note the need to visualize correlations between topics, as this can provide insight into what is happening on the document level without actually displaying entire documents. Last, we seek a solution

to the problem of visualizing a large number of topics (say, from 100 - 500 topics) in a compact way.

Adler, Daniel, Duncan Murdoch, and others. n.d. *Rgl: 3D Visualization Device System (OpenGL)*. <http://rgl.neoscientists.org>.

Alt, Alina, and Marvin S. White. 2008. Tracking an object with multiple asynchronous cameras. US 70219509. Patent, issued September 11, 2008. http://www.patentlens.net/patentlens/patent/US_7062320/.

Andreas Buja, Catherine Hurley, Daniel Asimov, and John A. McDonald. 1988. "Elements of a Viewing Pipeline for Data Analysis." In *Dynamic Graphics for Statistics*, edited by William S. Cleveland and Marylyn E. McGill. Belmont, California: Wadsworth, Inc.

Baumer, Ben, and Carson Sievert. n.d. *Etl: Extract-Transfer-Load Framework for Medium Data*. <http://github.com/beanumber/etl>.

Bischof, Jonathan M., and Edoardo M. Airolidi. 2012. "Summarizing Topical Content with Word Frequency and Exclusivity." In *Icml*.

Blei, David M., and John Lafferty. 2009. "Visualizing Topics with Multi-Word Expressions." *Arxiv*. <https://arxiv.org/pdf/0907.1013.pdf>.

Brynjar Gretarsson, Svetlin Bostandjiev, John O'Donovan, and Padhraic Smyth. 2011. "TopicNets: Visual Analysis of Large Text Corpora with Topic Modeling." In *ACM Transactions on Intelligent Systems and Technology*.

Buja, Andreas, Dianne Cook, Heike Hofmann, Michael Lawrence, Eun-Kyung Lee, Deborah F Swayne, and Hadley Wickham. 2009. "Statistical inference for exploratory

data analysis and model diagnostics.” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367 (1906): 4361–83.

Chambers, John. 1999. *Programming with Data*. Springer Verlag.

Chaney, Allison J.B., and David M. Blei. 2012. “Visualizing Topic Models.” In *Icwsm*.

Chang, Winston, and Hadley Wickham. 2015. *Ggvis: Interactive Grammar of Graphics*. <http://CRAN.R-project.org/package=ggvis>.

Chang, Winston, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2015. *Shiny: Web Application Framework for R*. <http://CRAN.R-project.org/package=shiny>.

Cheng, Joe. 2015a. *Crosstalk*. <https://github.com/rstudio/crosstalk>.

———. 2015b. *D3scatter: Demo of D3 Scatter Plot; Testbed for Crosstalk Library*. <https://github.com/jcheng5/d3scatter>.

Cheng, Joe, and Yihui Xie. 2015. *Leaflet: Create Interactive Web Maps with the Javascript ‘Leaflet’ Library*. <http://rstudio.github.io/leaflet/>.

Cook, Dianne, and Deborah F. Swayne. 2007. *Interactive and Dynamic Graphics for Data Analysis : With R and Ggobi*. Use R ! New York: Springer. <http://www.ggobi.org/book/>.

Daniel Ramage, Evan Rosen, Jason Chuang, Christopher D. Manning, and Daniel A. McFarland. 2009. “Topic Modeling for the Social Sciences.” In *NIPS 2009 Workshop on Applications for Topic Models: Text and Beyond*.

David M. Blei, Michael I. Jordan, Thomas L. Griffiths, and Joshua B. Tenenbaum. 2003. “Hierarchical Topic Models and the Nested Chinese Restaurant Process.” In *Nips*.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2012. “Latent Dirichlet Allocation.” In *Jmlr*.

David Mimno, Edmund Talley, Hanna M. Wallach, and Andrew McCallum. 2011. “Optimizing Semantic Coherence in Topic Models.” In *Emnlp*.

Visualization, and Interfaces, June, 1–8. <http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf>.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2016. *Plotly: Create Interactive Web-Based Graphs via Plotly's Api*. <https://github.com/ropensci/plotly>.

Swayne, Deborah F, Dianne Cook, and Andreas Buja. 1998. “XGobi: Interactive Dynamic Data Visualization in the X Window System.” *Journal of Computational and Graphical Statistics* 7 (1): 113–30.

Swayne, Deborah F., and Sigbert Klinke. 1999. “Introduction to the Special Issue on Interactive Graphical Data Analysis: What Is Interaction?” *Computational Statistics* 14 (1).

Taddy, Matthew A. 2011. “On Estimation and Selection for Topic Models.” In *AISTATS*.

Temple Lang, Duncan. 2014a. *RCurl: General Network (Http/Ftp/.) Client Interface for R*. <http://CRAN.R-project.org/package=RCurl>.

———. 2014b. *RJSONIO: Serialize R Objects to Json, Javascript Object Notation*. <http://CRAN.R-project.org/package=RJSONIO>.

Theus, Martin, and Simon Urbanek. 2008. *Interactive Graphics for Data Analysis: Principles and Examples*. Chapman & Hall / CRC.

Traub, James. 2010. “Mariano Rivera, King of the Closers.” <http://www.nytimes.com/2010/07/04/magazine/04Rivera-t.html>.

Unwin, Antony. 2006. “Exploratory Modelling Analysis: Visualizing the Value of Variables.” In *Compstat 2006 - Proceedings in Computational Statistics*, edited by

Alfredo Rizzi and Maurizio Vichi, 221–30. Physica-Verlag HD. http://dx.doi.org/10.1007/978-3-7908-1709-6_17.

Unwin, Antony, and Heike Hofmann. 2009. “GUI and Command-line - Conflict or Synergy?” *Proceedings of the St Symposium on the Interface*, September, 1–11.

Unwin, Antony, Chris Volinsky, and Sylvia Winkler. 2003. “Parallel Coordinates for Exploratory Modelling Analysis.” *Computational Statistics & Data Analysis* 43 (4): 553–64.

Urbanek, Simon. 2004. “Model Selection and Comparison Using Interactive Graphics.” PhD thesis.

Urbanek, Simon, and Jeffrey Horner. 2015. *FastRWeb: Fast Interactive Framework for Web Scripting Using R*. <http://CRAN.R-project.org/package=FastRWeb>.

Vaidyanathan, Ramnath. 2013. *RCharts: Interactive Charts Using Javascript Visualization Libraries*. <https://github.com/ramnathv/rCharts/>.

Vaidyanathan, Ramnath, Yihui Xie, JJ Allaire, Joe Cheng, and Kenton Russell. 2015. *Htmlwidgets: HTML Widgets for R*. <http://CRAN.R-project.org/package=htmlwidgets>.

Vanderkam, Dan, and JJ Allaire. 2015. *Dygraphs: Interface to Dygraphs Interactive Time Series Charting Library*. <http://CRAN.R-project.org/package=dygraphs>.

Wickham, Hadley. 2007. “Meifly: Models explored interactively.” *Website ASA Sections on Statistical Computing and Graphics (Student Paper Award Winner)*.

———. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. New York: Springer. <http://had.co.nz/ggplot2/book>.

———. 2014. “Tidy Data.” *The Journal of Statistical Software* 59 (10). <http://www.jstatsoft.org/v59/i10/>.

———. 2015a. *Httr: Tools for Working with Urls and Http*. <http://CRAN.R-project.org/package=httr>.