Ari: The Automated R Instructor

by Sean Kross, John Muschelli, Jeffrey T. Leek

Abstract We present the `ari` package for automatically generating technology-focused educational videos. The goal of the package is to be able to create reproducible videos, with the ability to change and update video content seamlessly. We present several examples of generating videos including using R Markdown slide decks, PowerPoint slides, or simple images as source material. We also discuss how `ari` can help instructors reach new audiences through programmatically translating materials into other languages.

Introduction

Videos are a crucial way people learn and they are pervasive in online education platforms (TODO: Cite). Producing educational videos with a lecturer speaking over slides takes time, energy, and usually video editing skills. Maintaining the accuracy and relevancy of lecture videos focused on technical subjects like computing programming or data science can often require remaking an entire video, requiring extensive editing and splicing of new segments. We present **ari**, the **A**utomated **R** **I**nstructor as a tool to address these issues by creating reproducible presentations and videos that can be automatically generated from plain text files or similar artifacts. By using **ari**, we provide a tool for users to rapidly create and update video content.

In its simplest form a lecture video is comprised of visual content (e.g. slides and figures) and a spoken explanation of the visual content. Instead of a human lecturer the **ari** package uses a text-to-speech system to synthesize spoken audio for a lecture. Modern text-to-speech systems which take advantage of recent advancements in artificial intelligence research are available from Google, Microsoft, and Amazon. Many of these synthesizers make use of deep learning methods, such as WaveNet (Van Den Oord et al. 2016) and have interfaces in R (Edmondson 2019; Muschelli 2019a; Leeper 2017). Currently in **ari**, synthesis of the the audio can be rendered using any of these services through the **text2speech** (Muschelli 2019b) package. The default is Amazon Polly, which has text to speech voice generation in over 21 languages, including a total of 29 dialects, implemented in the **aws.polly** package (Leeper 2017). In addition to multiple languages, the speech generation services provide voices with several pitches representing different genders within the same language. We present the **ari** package with reproducible use case examples and the video outputs with different voices in multiple languages.

The **ari** package relies on the **tuneR** package for splitting and combining audio files appropriately so that lecture narration is synced with each slide (Ligges et al. 2018). Once the audio is generated, it is synced with images to make a lecture video. Multiple open source tools for video editing and splicing exist. **ari** takes advantage of the `ffmpeg` (`http://www.ffmpeg.org/`) software, a command-line interface to the `libav` library. These powerful tools have been thoroughly tested with a development history spanning almost 20 years. **ari** uses `ffmpeg` to piece together images and audio into a lecture video. **ari** has been built with presets for `ffmpeg` which allow output videos to be compatible with multiple platforms, including the YouTube and Coursera players. These presets include specifying the bitrate, audio and video codecs, and the output video format. The numerous additional video specifications for customization can be applied to command-line arguments `ffmpeg` through **ari**.

We have developed a workflow with **ari** as the centerpiece for automatically generating educational videos. The narration script for lecture videos is stored in a plain text format, so that it can be synthesized into audio files via text-to-speech services. By storing lecture narration in plain text it can be updated, tracked, and collaboratively or automatically updated with version control software like Git and GitHub. If the figures in the lecture slides are created in a reproducible framework, such as generated using R code, the entire video can be reproducibly created and automatically updated. Thus, **ari** is the Automated R Instructor. We will provide examples of creating videos based on the following sets of source files: a slide deck built with R Markdown, a set of images and a script, or a Google Slide deck or PowerPoint presentation.

Making videos with `ari`: `ari_stitch`

The main workhorse of **ari** is the `ari_stitch` function. This function requires an ordered set of images and an ordered set of **tuneR** audio objects that correspond to each image. The `ari_stitch` function sequentially "stitches" each image which is shown in the video for the duration of its corresponding audio object using `ffmpeg`. In order to use **ari**, one must have an `ffmpeg` installation to combine the audio and images. In the example below, 2 images (packaged with **ari**) are overlaid withe white noise for demonstration. This example also allows users to check if the output of `ffmpeg` works with a desired video player.

```
library(tuneR)
library(ari)
result = ari_stitch(
  ari_example(c("mab1.png", "mab2.png")),
  list(noise(), noise()),
  output = "noise.mp4")
isTRUE(result)
attributes(result)$outfile
```

```
[1] TRUE
```

```
[1] "noise.mp4"
```

The output is a logical indicator of success and the path of the output file. The video for this output can be seen at https://youtu.be/3kgaYf-EV90.

Synthesizer authentication

The above example uses `tuneR::noise()` to generate audio and to show that any audio object can be used with **ari**. In most cases however, **ari** is most useful when combined with synthesizing audio using a text-to-speech system. Though one can generate the spoken audio in many ways, such as fitting a custom deep learning model, we will focus on using the aforementioned services (e.g. Amazon Polly) as they have straightforward public web APIs. One obstacle in using such services is that users must go through steps to provide authentication, whereas most of these APIs and the associated R packages do not allow for interactive authentication such as OAuth.

The **text2speech** package provides a unified interface to these 3 text-to-speech services, and we will focus on Amazon Polly and its authentication requirements. Polly is authenticated using the **aws.signature** package (Leeper 2019). The **aws.signature** documentation provides options and steps to create the relevant credentials; we have also provided an additional tutorial. Essentially, the user must sign up for the service and retrieve public and private API keys and put them into their R profile or other areas accessible to R. Running `text2speech::tts_auth(service = "amazon")` will indicate if authentication was successful (if using a different service, change the `service` argument). NB: The APIs are generally paid services, but many have free tiers or limits, such as Amazon Polly's free tier for the first year (https://aws.amazon.com/polly/pricing/).

Creating Speech from Text: `ari_spin`

After Polly has been authenticated, videos can be created using the `ari_spin` function with an ordered set of images and a corresponding ordered set of text strings. This text is the "script" that is spoken over the images to create the output video. The number of elements in the text needs to be equal to the number of images. Let us take a part of Mercutio's speech from Shakespeare's Romeo and Juliet (Shakespeare 2003) and overlay it on two images from the Wikipedia page about Mercutio (https://en.wikipedia.org/wiki/Mercutio):

```
speech =  c(
  "I will now perform part of Mercutio's speech from Shakespeare's Romeo and Juliet.",
  "O, then, I see Queen Mab hath been with you.
   She is the fairies' midwife, and she comes
   In shape no bigger than an agate-stone
   On the fore-finger of an alderman,
   Drawn with a team of little atomies
   Athwart men's noses as they lies asleep;")
mercutio_file = "death_of_mercutio.png"
mercutio_file2 = "mercutio_actor.png"

shakespeare_result = ari_spin(
  c(mercutio_file, mercutio_file2),
  speech, output = "romeo.mp4", voice = "Joanna")
isTRUE(shakespeare_result)
```

```
[1] TRUE
```

The speech output can be seen at https://youtu.be/SFhvM9gI0kE . We chose the voice "Joanna" which is designated as a female sounding US-English speaker for the script. Each voice is language-dependent; we can see the available voices for English for Amazon Polly at https://docs.aws.amazon.com/polly/latest/dg/SupportedLanguage.html.

Though the voice generation is relatively clear, we chose a Shakespearean example to demonstrate the influence and production value of the variety of dialects available from these text-to-speech services. Compare the video of "Joanna" to the same video featuring "Brian" who "speaks" with a British English dialect:

```
gb_result = ari_spin(
  c(mercutio_file, mercutio_file2),
  speech, output = "romeo_gb.mp4", voice = "Brian")
isTRUE(gb_result)
```

```
[1] TRUE
```

The resulting video can be seen at https://youtu.be/fSS0JSb4VxM.

The output video format is MP4 by default, but several formats can be specified via specifying the appropriate "muxer" for `ffmpeg` (see the function `ffmpeg_muxers`). Supported codecs can be founded using the functions `ffmpeg_audio_codecs` and `ffmpeg_video_codecs`.

We now discuss the number of image and script inputs that **ari** is designed to work with, including text files and a series of PNG images, a Google Slide deck or a PowerPoint presentation with the script written in the speaker notes section, or an HTML slide presentation created from an R Markdown, where the script is written in the HTML comments.

Creating Videos from R Markdown Documents

Many R users have experience creating slide decks with R Markdown, for example using the **rmarkdown** or **xaringan** packages (Allaire et al. 2019; Xie, Allaire, and Grolemund 2018; Xie 2018). In **ari**, the HTML slides are rendered using **webshot** (Chang 2018) and the script is located in HTML comments (i.e. between `<!--` and `-->`). For example, in the file `ari_comments.Rmd`, which is an `ioslides` type of R Markdown slide deck, we have the last slide:

```
x = readLines(ari_example("ari_comments.Rmd"))
tail(x[ x != ""], 4)
```

```
[1] "## Conclusion"
[2] "<!--"
[3] "Thank you for watching this video and good luck using Ari!"
[4] "-->"
```

so that the script for this slide starts with `"Thank you"`. This setup allows for one plain text, version-controllable, integrated document that can reproducibly generate a video. We believe these features allow creators to make agile videos, that can easily be updated with new material or changed when errors or typos are found.

Users can pass in both the R Markdown document and the rendered slides from the document, or simply the document, and the output will be created using `render` from **rmarkdown** (Allaire et al. 2019). Here we create the video for `ari_comments.Rmd`:

```
# Create a video from an R Markdown file with comments and slides
res = ari_narrate(
  script = ari_example("ari_comments.Rmd"),
  voice = "Kendra",
  capture_method = "iterative")
```

The output video is located at https://youtu.be/rv9fg_qsqc0. In our experience with several users we have found that some HTML slides take more or less time to render when using **webshot**; for example they may be tinted with gray because they are in the middle of a slide transition when the image of the slide is captured. Therefore we provide the `delay` argument in `ari_narrate` which is passed to **webshot**. This can resolve these issues by allowing more time for the page to fully render, however this means it may take for more time to create each video. We also provide the argument `capture_method` to allow for finely-tuned control of `webshot`. When `capture_method = "vectorized"`, **webshot** is run on the entire slide deck in a faster process, however we have experienced slide rendering issues with this setting depending on the configuration of an individual's computer.

However when `capture_method = "iterative"`, each slide is rendered individually in `webshot`, which solves many rendering issues, however it causes videos to be rendered more slowly.

With respect to accessibility, **ari** encourages video creators to type out a script by design. This provides an effortless source of subtitles for people with hearing loss rather than relying on other services, such as YouTube, to provide speech-to-text subtitles. When using `ari_spin`, if the `subtitles` argument is `TRUE`, then an SRT file for subtitles will be created with the video.

One issue with synthesis of technical information is that changes to the script are required for Amazon Polly or other services to provide a correct pronunciation. For example, if you want the service to say "RStudio" or "ggplot2", the phrases "R Studio" or "g g plot 2" must be written exactly that way in the script. These phrases will then appear in an SRT subtitle file, which may be confusing to a viewer.

Creating Videos from Other Documents

In order to create a video from a Google Slide deck or PowerPoint presentation, the slides should be converted to a set of images. We recommend using the PNG format for these images. In order to get the script for the video, we suggest putting the script for each slide in the speaker notes section of that slide. Several of the following features for video generation are in our package **didactr** (`https://github.com/muschellij2/didactr`). The speaker notes of slides can be extracted using **rgoogleslides** (Noorazman 2018) for Google Slides via the API or using **readOffice**/**officer** (Gohel 2019; Ewing 2017) to read from PowerPoint documents. Google Slides can be downloaded as a PDF and converted to PNGs using the **pdftools** package (Ooms 2019). The **didactr** package also has a `pptx_notes` function for reading PowerPoint notes. Converting PowerPoint files to PDF can be done using LibreOffice and the **docxtractr**(Rudis and Muir 2019) package which contains the necessary wrapper functions.

To demonstrate this, we use an example PowerPoint is located on Figshare (`https://figshare.com/articles/Example_PowerPoint_for_ari/8865230`). We can convert the PowerPoint to PDF, then to a set of PNG images, then extract the speaker notes.

```
pptx = "ari.pptx"
pdf = docxtractr::convert_to_pdf(pptx)
pngs = pdftools::pdf_convert(pdf, dpi = 300)
notes = didactr::pptx_notes(pptx)
notes

[1] "Sometimes it's hard for an instructor to take the time to record their lectures.
For example, I'm in a coffee shop and it may be loud."

[2] "Here is an example of a plot with really small axes.  We plot the x versus the y
-variables and a smoother between them."
```

We can then render the video with the "Kimberly" voice. We use the `divisible_height` argument to forcibly scale the height of the images to be divisible by 2 if necessary. This is required by the`x264` codec which we have specified as a preset:

```
pptx_result = ari_spin(pngs, notes, output = "pptx.mp4", voice = "Kimberly",
    divisible_height = TRUE, subtitles = TRUE)
isTRUE(pptx_result)
```

You can see the output at `https://youtu.be/TBb3Am6xsQw`. Here we can see the first few lines of the subtitle file:

```
[1] "1"
[2] "00:00:00,000 --> 00:00:02,025"
[3] "Sometimes it's hard for an instructor to"
[4] "2"
[5] "00:00:02,025 --> 00:00:04,005"
[6] "take the time to record their lectures."
```

For Google Slides, the slide deck can be downloaded as a PowerPoint and the previous steps can be used, however it can also be downloaded directly as a PDF. The **didactr** package has the function `gs_notes_from_slide` to extract the notes for synthesis. As this extraction process requires authentication, we will omit it here. Thus, we should be able to create videos using R Markdown, Google Slides, or PowerPoint presentations in an automatic fashion.

Summary

The **ari** package combines multiple open-source tools and APIs to create reproducible workflows for creating videos. These videos can be created using R Markdown documents, PowerPoint presentations, Google Slide decks, or simply series of images. The audio overlaid on the images can be separate or contained within the storage of the images. These workflows can then be reproduced in the future and easily updated. As the current voice synthesis options are somewhat limited in the tenacity and inflection given, we believe that educational and informational videos are the most applicable area.

Future directions

ari and **didactr** are already being used to build data science curriculla (Kross and Guo 2019) and we look forward to collaborating with video creators to augment **ari** according to their changing needs. In the following section we outline possible directions for the future of the project.

Since **ari** is designed for teaching technical content, we plan to provide better support for the pronunciation of technical terms like the names of popular software tools. These names are usually not pronunced correctly by text-to-speech services because they are not words contained in the training data used in the deep learning models used by these services. To address this concern we plan to compile a dictionary of commonly used technical terms and the phonetic phrasing and spelling of these terms that are required in order to achieve the correct pronounciation from text-to-speech services.

In addition to still images and synthesized voices, we would like to develop new technologies for incorperating other automatically generated videos into lectures generated by **ari**. As computer programming, statistics, and data science instructors we often rely on live coding (Chen and Guo 2019) to demonstrate software tools to our students. Live coding videos suffer from many of the same problems as other kinds of technical videos as we addressed in the introduction. Therefore we plan to build a system for automating the creation of live coding videos. These videos would also be created using plain text documents like R Markdown. They would integrate synthesized narration with code chunks that would be displayed and executed according to specilized commands that would specify when code should be executed in an IDE like RStudio. These commands could also control which panes and tabs of the IDE are visible or emphasized.

As programmatic video creation software improves, we plan to extend **ari** so it can expand its compatibility with different technologies. For example we believe the heavy reliance on an `ffmpeg` installation can be mitigated in the future with advances in the **av** package. Though the **av** package has powerful functionality and is currently porting more from `libav` and therefore `ffmpeg`, it currently does not have the capabilities required for **ari**. Although third party installation from `https://ffmpeg.org/` can be burdensome to a user, package managers such as `brew` for OSX and `choco` for Windows provide an easier installation and configuration experience.

Although we rely on Amazon Polly for voice synthesis, other packages provide voice synthesis, such as **mscstts** for Microsoft and **googleLanguageR** for Google. We created the **text2speech** package to harmonize these synthesis options for **ari**.

We see significant potential in how **ari** could expand global learning opportunities. Video narration scripts can be automatically translated into other languages with services like the Google Translation API, where **googleLanguageR** provides an interface. Amazon Polly can speak languages other than English, meaning that one can write a lecture once and generate lecture videos in multiple languages. Therefore this workflow can greatly expand the potential audience for educational videos with relatively little additional effort from lecture creators. We plan to flesh out these workflows so that video creators can manage videos in multiple languages. We hope to add functionality so that communities of learners with language expertise can easily suggest modifications to automatically translated videos, and tooling so suggestions can be incorperated quickly.

The workflow we have specified requires many interlocking pieces of software, therefore we have created a Docker environment (`https://github.com/seankross/bologna`) which contains the required packages to create videos using **ari**. This Docker image ensures that our workflow is completely reproducable, and it also enables deployment to multiple disk images for creating lecture videos at scale.

Conclusions

The **ari** package combines multiple open-source tools and APIs to create reproducible workflows for creating educational lecture videos. These videos can be created using R Markdown documents, PowerPoint presentations, Google Slide decks, or simply a series of images. The source of the audio overlaid on the images can be stored in a separate file or it can be contained within any of the specified file formats. Our workflow ensures thats these videos can then be reproduced in the future and easily updated. Given the content that it is practical to create with this workflow, and the limtted nature of voice synthesis tools, we believe that **ari** is most appropriate for the production of educational and informational videos.

Allaire, JJ, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2019. *rmarkdown: Dynamic Documents for R*. https://rmarkdown.rstudio.com.

Chang, Winston. 2018. *webshot: Take Screenshots of Web Pages*. https://CRAN.R-project.org/package=webshot.

Chen, Charles, and Philip J. Guo. 2019. "Improv: Teaching Programming at Scale via Live Coding." In *Proceedings of the Sixth Annual Acm Conference on Learning at Scale*. L@S '19. New York, NY, USA: ACM. https://doi.org/10.1145/3330430.3333627.

Edmondson, Mark. 2019. *googleLanguageR: Call Google's 'Natural Language' API, 'Cloud Translation' API, 'Cloud Speech' API and 'Cloud Text-to-Speech' API*.

Ewing, Mark. 2017. *readOffice: Read Text Out of Modern Office Files*. https://CRAN.R-project.org/package=readOffice.

Gohel, David. 2019. *officer: Manipulation of Microsoft Word and PowerPoint Documents*. https://CRAN.R-project.org/package=officer.

Kross, Sean, and Philip J. Guo. 2019. "End-User Programmers Repurposing End-User Programming Tools to Foster Diversity in Adult End-User Programming Education." In *Proceedings of VL/HCC 2019: IEEE Symposium on Visual Languages and Human-Centric Computing*. VL/HCC '19. Memphis, TN, USA.

Leeper, Thomas J. 2017. *aws.polly: Client for AWS Polly*.

———. 2019. *aws.signature: Amazon Web Services Request Signatures*.

Ligges, Uwe, Sebastian Krey, Olaf Mersmann, and Sarah Schnackenberg. 2018. *tuneR: Analysis of Music and Speech*. https://CRAN.R-project.org/package=tuneR.

Muschelli, John. 2019a. *mscstts: R Client for the Microsoft Cognitive Services 'Text-to-Speech' REST API*. https://CRAN.R-project.org/package=mscstts.

———. 2019b. *text2speech: Text to Speech*. https://github.com/muschellij2/text2speech.

Noorazman, Hairizuan Bin. 2018. *rgoogleslides: R Interface to Google Slides*. https://CRAN.R-project.org/package=rgoogleslides.

Ooms, Jeroen. 2019. *pdftools: Text Extraction, Rendering and Converting of PDF Documents*. https://CRAN.R-project.org/package=pdftools.

Rudis, Bob, and Chris Muir. 2019. *docxtractr: Extract Data Tables and Comments from Microsoft Word Documents*. http://gitlab.com/hrbrmstr/docxtractr.

Shakespeare, William. 2003. *Romeo and Juliet*. Cambridge University Press.

Van Den Oord, Aäron, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. 2016. "WaveNet: A Generative Model for Raw Audio." *SSW* 125.

Xie, Yihui. 2018. *xaringan: Presentation Ninja*. https://CRAN.R-project.org/package=xaringan.

Xie, Yihui, J.J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown.

Sean Kross
Cognitive Science, University of California, San Diego
9500 Gilman Dr.
La Jolla, CA 92093
seankross@ucsd.edu

John Muschelli
Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health
615 N Wolfe Street
Baltimore, MD 21231
jmusche1@jhu.edu

Jeffrey T. Leek
Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health
615 N Wolfe Street
Baltimore, MD 21231

jtleek@jhu.edu