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 discusss how `ari` can help instructors reach new audiences through programmatically translating materials into other languages.

Introduction

Videos are a crucial way people learn and 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. In leiu (TODO: check usage) of a human lecturer the **ari** package uses a text-to-speech system to

you have visual content (e.g. slides, figures) and you want to explain them with words (i.e. a script) in a video. Voice synthesizer services are available from Google, Microsoft, and Amazon. Many of these synthesizers take 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** package (Muschelli 2019b). 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 of 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 reading and manipulating audio output to combine split audio files and to add pauses to audio files between slides (Ligges et al. 2018). Once the audio is generated, it much be spliced with the images to make the video. Multiple open source tools for video editing and splicing exist. The `ffmpeg` (`http://www.ffmpeg.org/`) software is highly powerful, has been thoroughly tested, and has been developed for almost 20 years; **ari** uses `ffmpeg` to overlay the images over the audio. The output videos have been tested on multiple platforms, including the YouTube and Coursera players. A default specification is used in **ari**, such as bitrate, audio and video codecs used, and output video format. The numerous additional video specifications can be applied to command-line arguments `ffmpeg` through **ari**.

With these tools together, we can generate automated content; we have used **ari** for educational videos. The spoken scripts for these videos can be stored in plain text, and therefore be version controlled, edited, and updated easily. If the figures 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 a slide deck in R Markdown, a set of images and a script, and discuss how to create slides using 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 the audio to overlay on some images to have already been generated. The `ari_stitch` function takes audio and images, and "stitches" them together 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 with 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

In most cases, however, we do must generate the audio to overlay on images. Though one can generate the spoken audio in many ways, such as fitting a custom deep learning model, we will use the aforementioned services (e.g. Google) as they have direct APIs for use. The downside of 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 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 accesssible 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 using the `ari_spin` function with a set of images and of text. This text is the "script" that is spoken over the images to create the output video. The number of elements in the text need 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 2 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)
```

The speech output can be seen at https://youtu.be/SFhvM9gI0kE . We chose the voice "Joanna" to the the female US-English speaker for the script. The voices are language-dependent; we can see the available voices for English for Amazon Polly below (from https://docs.aws.amazon.com/polly/latest/dg/SupportedLanguage.html):

voice	language	language_code	gender	service
Russell	Australian English	en-AU	Male	amazon
Nicole	Australian English	en-AU	Female	amazon
Amy	British English	en-GB	Female	amazon
Brian	British English	en-GB	Male	amazon
Emma	British English	en-GB	Female	amazon
Raveena	Indian English	en-IN	Female	amazon
Aditi	Indian English	en-IN	Female	amazon
Salli	US English	en-US	Female	amazon
Joanna	US English	en-US	Female	amazon
Matthew	US English	en-US	Male	amazon
Ivy	US English	en-US	Female	amazon
Justin	US English	en-US	Male	amazon
Kendra	US English	en-US	Female	amazon
Kimberly	US English	en-US	Female	amazon
Joey	US English	en-US	Male	amazon
Geraint	Welsh English	en-GB-WLS	Male	amazon

Though the voice generation is relatively clear, we would not classify the speech as passionate or with a high level of emphasis. Thus, be believe these videos may be best used for conveying information for education as opposed to entertainment. We can also generate the video using the voice `Brian`, which is an British English male voice:

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

The speech output can be seen at https://youtu.be/fSS0JSb4VxM. The output video format is MP4 by default, but can be any format (aka "muxers") that the `ffmpeg` installation support, see the function `ffmpeg_muxers`. Supported codecs can be founded using the functions `ffmpeg_audio_codecs` and `ffmpeg_video_codecs`. The images and script can be presented in a number of ways, such as a text file and a series of PNG images. More likely, the images and script will be bundled together, such as a Google Slide deck/PowerPoint presentation with the script in the notes section, or an HTML slide presentation based in R Markdown, where the script is in the HTML comments.

Creating Videos from R Markdown Documents

For most R users, we believe the most natural setting is that the user has a slide deck using 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 `ari_comments.Rmd`, which is a `ioslides` type of 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 to be spoken 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 resulting output, 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. Some HTML slides take a bit to render on **webshot**; for example may be rendered dark gray instead of white. If you change the `delay` argument in `ari_narrate`, passed to **webshot**, this can resolve some issues by allowing the page to fully render, but may take a bit longer to run. Also, the argument `capture_method` allows for the control on how `webshot` is run. Using the value `vectorized`, **webshot** is run on the entire slide deck and is faster, but may have some issues. The value `iterative` runs `webshot` for each slide separately, which can be more robust, but can be slower. These issues are negligible when there are only a few slides such as in these examples, but are relevant when the number of slides increases.

With respect to accessibility, as **ari** has the synthesized script, this provides for direct subtitles for those hard of hearing rather than relying on other services, such as YouTube, to provide a speech to text translation. When using `ari_spin`, if the `subtitles` flag is marked true, then an SRT file will be output 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 script should say "R Studio" and "g g plot 2". Thus, you may want to make edits to the subtitle file before uploading.

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, likely PNGs. In order to get the script for the video, we suggest putting the script for each slide in the notes section of that slide. We have built some of this additional functionality for video generation in our package **didactr** (https://github.com/muschellij2/didactr). The 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 PDF and converted to PNGs using the **pdftools** package (Ooms 2019). The **didactr** package also has a `pptx_notes` function for reading PowerPoint notes and wraps most of the functionality for conversion. Converting from PowerPoint to PDF can be done using LibreOffice, which **docxtractr**, versions `0.6.2` and above (Rudis and Muir 2019) has wrapper functions to achieve this.

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 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 ensure the height of the images are divisible by 2, as the `x264` codec we are using requires this:

```
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; 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

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 installations.

Although we rely on Amazon Polly for voice synthesis, other packages provide voice synthesis, such as **mscstts** for Microsoft and **googleLanguageR** for Google.

Scripts can be automatically translated into other languages with services like the Google Translation API, which **googleLanguageR** provides an interface. Amazon Polly can speak languages other than English. This means you can write a lecture once and generate slides and videos in multiple languages.

We have created a Docker environment (https://github.com/seankross/bologna) with the requirements to create videos using **ari**. This Docker image increases the level of reproducibility and can be used to create standalone disk images to create content.

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.

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.

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