The Automated R Instructor

by Sean Kross, John Muschelli, Jeffrey T. Leek

Abstract An abstract of less than 150 words.

Introduction

Videos are a large way people learn. Creating videos of an speaker with slides take time, energy, and usually a bit if video editing skills. A large issue with such videos is that updating the materials either requires remaking the entire video or extensive editing and splicing of new segments. We present **ari**, the automated R instructor to mitigate these issues by creating reproducible presentations and videos that can be automatically generated. By using Ari we hope to be able to rapidly create and update video content.

Multiple open source tools for video editing and splicing exist. The `ffmpeg` software is highly powerful, has been thoroughly tested, and has been developed for over XX years.

Technical stuff

The **ari** package relies on FFmpeg (>= 3.2.4) to interleave the images and the audio files.

Amazon authetication

The **ari** package relies on the **tuneR** package for reading and manipulating audio files such as MP3 and WAV files. The voice synthesis is done by Amazon Polly, which is a text to speech voice generation engine with over XX languages, implemented in the **aws.polly** package. In addition to multiple languages, Amazon Polly provides voices of different gender within the same language, as well as different dialects such as American English.

In order to use Amazon Polly, you must set up R to use Amazon Web Services, which we have provided as a tutorial here. In order to test the authentication, you can run `aws.polly::list_voices()`.

Making videos with `ari`

Videos can be produced form `ari` using an HTML slide presentation based in RMarkdown, where the script is in the HTML comments, or a simple vector of images and paragraphs.

The main function is `ari_spin`, which takes in a set of images and a series of paragraphs of text. These paragraphs are the "script" that is spoken over the images to create the output video. The number of paragraphs need to be equal to the number of images. The output video format is MP4 by default, but can be any format that the `ffmpeg` installation codecs support. Supported codecs can be founded using the functions `ffmpeg_audio_codecs` and `ffmpeg_video_codecs`.

The main workhorse of **ari** is the `ari_stitch` function. This function takes in a vector of images and a series of Wav audio objects or audio filenames.
We have tested the videos on YouTube and the Coursera platform. Additional video specifications can be applied to `ffmpeg_opts` argument of `ari_stitch`.

Accessibility

With respect to accessibility, as **ari** has the direct script that was spoken, 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. Though some changes to the script are required for AMazon Polly to correctly pronounce the information, these can be changed using regular expressions in the script, and then passed to `ari_subtitles`.

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 requried 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. We aim to harmonize these synthesis options, so that users can choose to create videos with the services that they support or have access to.

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.

Examples (FROM README - edit)

These examples make use of the `ari_example()` function. In order to view the files mentioned here you should use `file.show(ari_example("[file name]"))`. You can watch an example of a video produced by Ari here.

```
library(ari)

# First set up your AWS keys
Sys.setenv("AWS_ACCESS_KEY_ID" = "EA6TDV7ASDE9TL2WI6RJ",
           "AWS_SECRET_ACCESS_KEY" = "OSnwITbMzcAwvHfYDEmk10khb3g82j04Wj8Va4AA",
           "AWS_DEFAULT_REGION" = "us-east-2")

# Create a video from a Markdown file and slides
ari_narrate(
  ari_example("ari_intro_script.md"),
  ari_example("ari_intro.html"),
  voice = "Joey")

# Create a video from an R Markdown file with comments and slides
ari_narrate(
  ari_example("ari_comments.Rmd"),
  ari_example("ari_intro.html"),
  voice = "Kendra")

# Create a video from images and strings
ari_spin(
  ari_example(c("mab1.png", "mab2.png")),
  c("This is a graph.", "This is another graph"),
  voice = "Joanna")

# Create a video from images and Waves
library(tuneR)
ari_stitch(
  ari_example(c("mab1.png", "mab2.png")),
  list(noise(), noise()))
```

RMarkdown/HTML slide Problems

Some html slides take a bit to render on webshot, and can be dark gray instead of white. If you change the `delay` argument in `ari_narrate`, passed to `webshot`, this can resolve some issues, but may take a bit longer to run. Also, using `capture_method = "vectorized"` is faster, but may have some issues, so run with `capture_method = "iterative"` if this is the case as so:

```
ari_narrate(
  ari_example("ari_comments.Rmd"),
  ari_example("ari_intro.html"),
  voice = "Kendra",
  delay = 0.5,
  capture_method = "iterative")
```

Sean Kross
UCSD
line 1

line 2
author1@work

John Muschelli
Affiliation
line 1
line 2
author2@work

Jeffrey T. Leek
Affiliation
line 1
line 2
author2@work