Making Magic with Keras and Shiny

An exploration of Shiny's position in the data science pipeline

Nick Strayer

2018/01/22

Me

- @NicholasStrayer
- nickstrayer.me
- LiveFreeOrDichotomize

- PhD Candidate in Biostatistics at Vanderbilt University
- Previously:
  - Johns Hopkins Data Science Lab
  - New York Times graphics department
  - Data visualization "engineer"

Slides

nickstrayer.me/dataDayTexas

Outline

Three sections

1. What...
2. Why...
3. How...

... I did

What?

Shiny app for casting spells

source

> Any sufficiently advanced technology is indistinguishable from magic.

- Arthur C. Clark

Demo

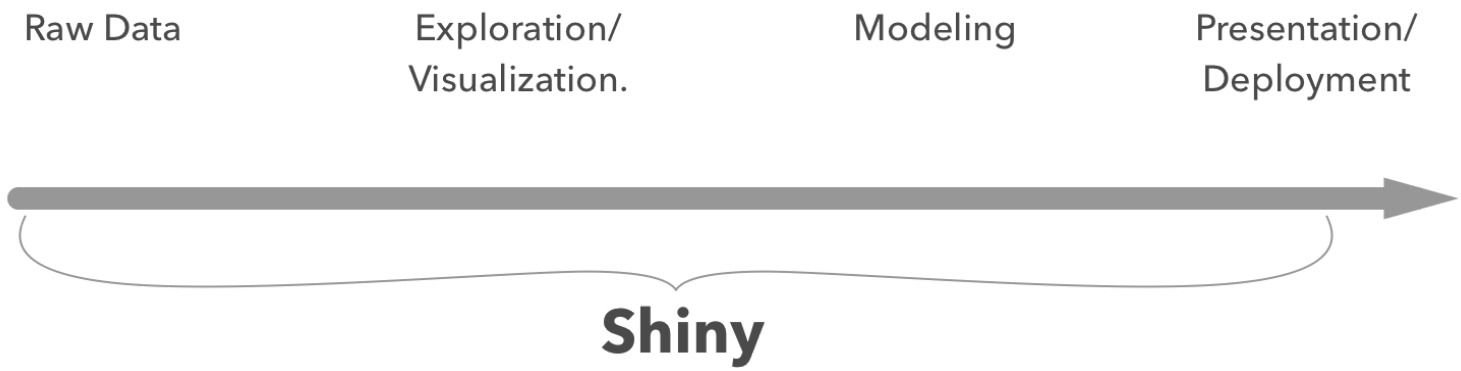
In which the rest of the talk is invalidated...

Three main steps:

1. Shiny app for gathering data
2. Modeling data with Keras
3. Shiny app for presenting data

Why?

Data Science Progression



I want to show Shiny can fit into the datascience workflow from the begining to the end.

Traditionally shiny has been used to show off results afterwards, but that's missing a huge portion of its potential.

Shiny for gathering data

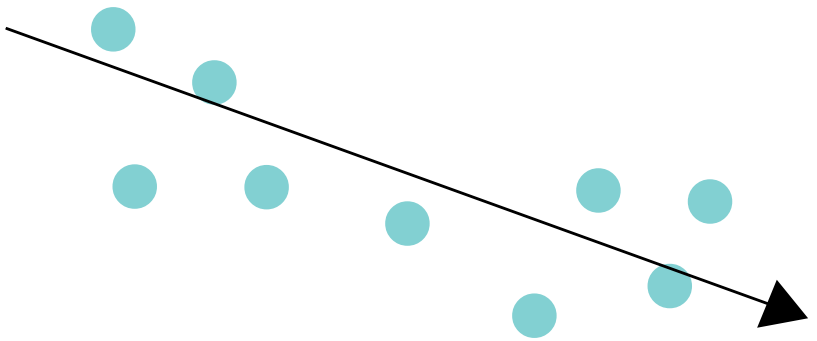
Who is better at knowing what they want their data to look like more than the ones who are going to be using it?

source

Many people like statisticians are given entire courses on how to set up things like clinical trials, but datascience often assumes the data has magically appeared.

Shiny for presenting models

This is the traditional way shiny is used, and there's a reason for that: it's great.



Instead of having to get fancy with learning how to port tensorflow models to Swift or Java you can just throw a predict function inside of a simple app.

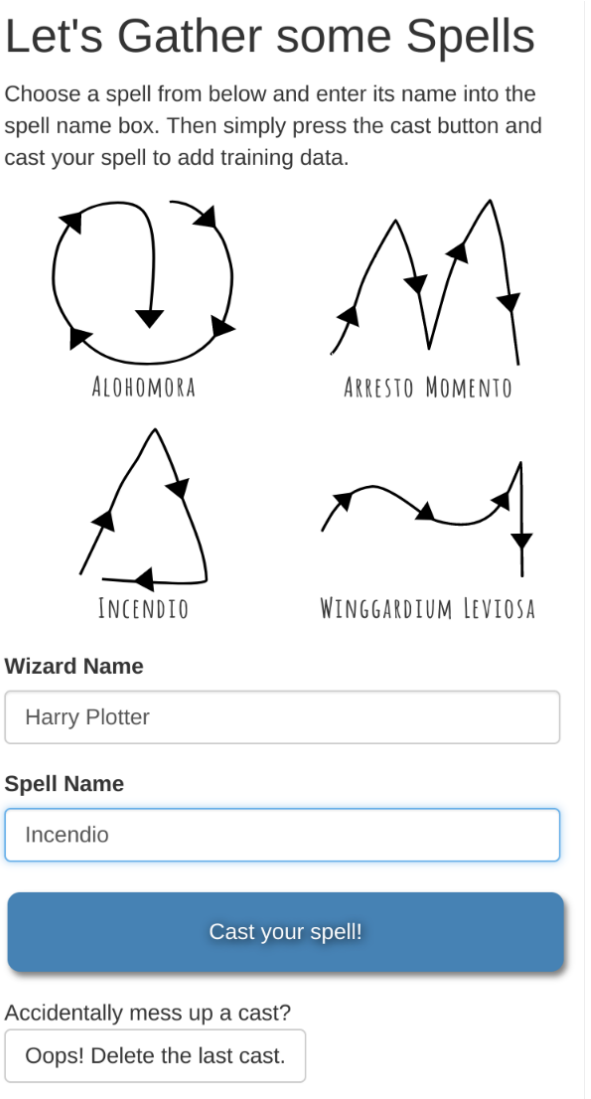
How well can this scale?

How?

Data gathering app

We need to...

- Gather data from accelerometer.
- Visualize results so that I could make sure I wasn't recording bad data.
- Save data somewhere I could use it.



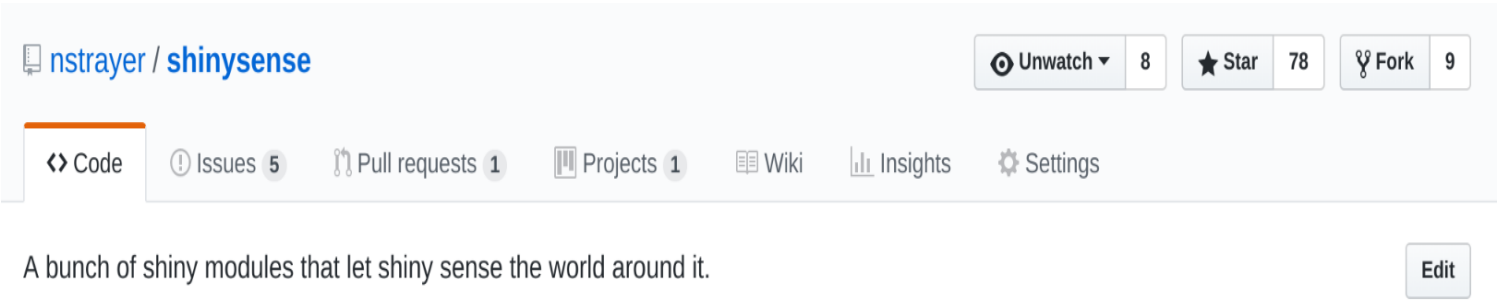
Shiny

> Shiny is an R package that makes it easy to build interactive web apps straight from R.

RStudio

Gathering data from accelerometer

Used my package `shinysense`.



Contains functions to bring data into Shiny apps.

One of these data sources happens to be accelerometer data.

Movr

ui.R

```
shinymovrUI("movr_button")
```

server.R

```
movement <- callModule(
  shinymovr,
  "movr_button")
```

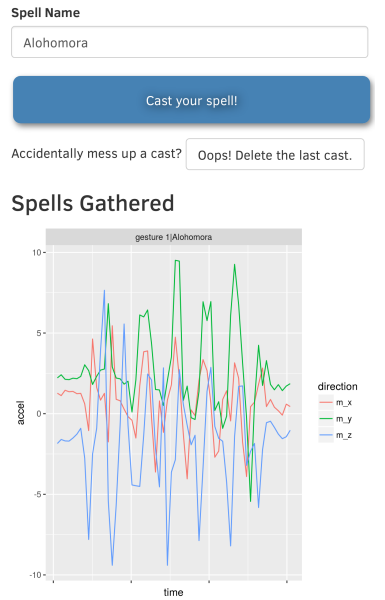
Binds to the `devicemotion` api in javascript to pull in motion data.

```
window.addEventListener("deviceorientation", handleMovementFunc);
```

Visualizing data

Not that hard as `shinysense::movr` returns an observable that can be used like any other shiny object.

```
movement() %>%
  ggplot(aes(
    x = time,
    y = accel
  )) +
  geom_line(aes(
    color = direction
  )) + ...
```



Added a delete button if bad data was detected.

Saving the data

Two options:

1. Use shinyapps.io and do a dropbox or similar upload
2. Just host the app from my own server
    - This was easier.
    - Better for other reasons I will get to later as well.

Luckily the RStudio Server makes this crazy easy.

```
write_csv(rvs$movements, paste0(sessionId, '_movement.csv'))
```

Data exploration/ visualization

Getting data all in

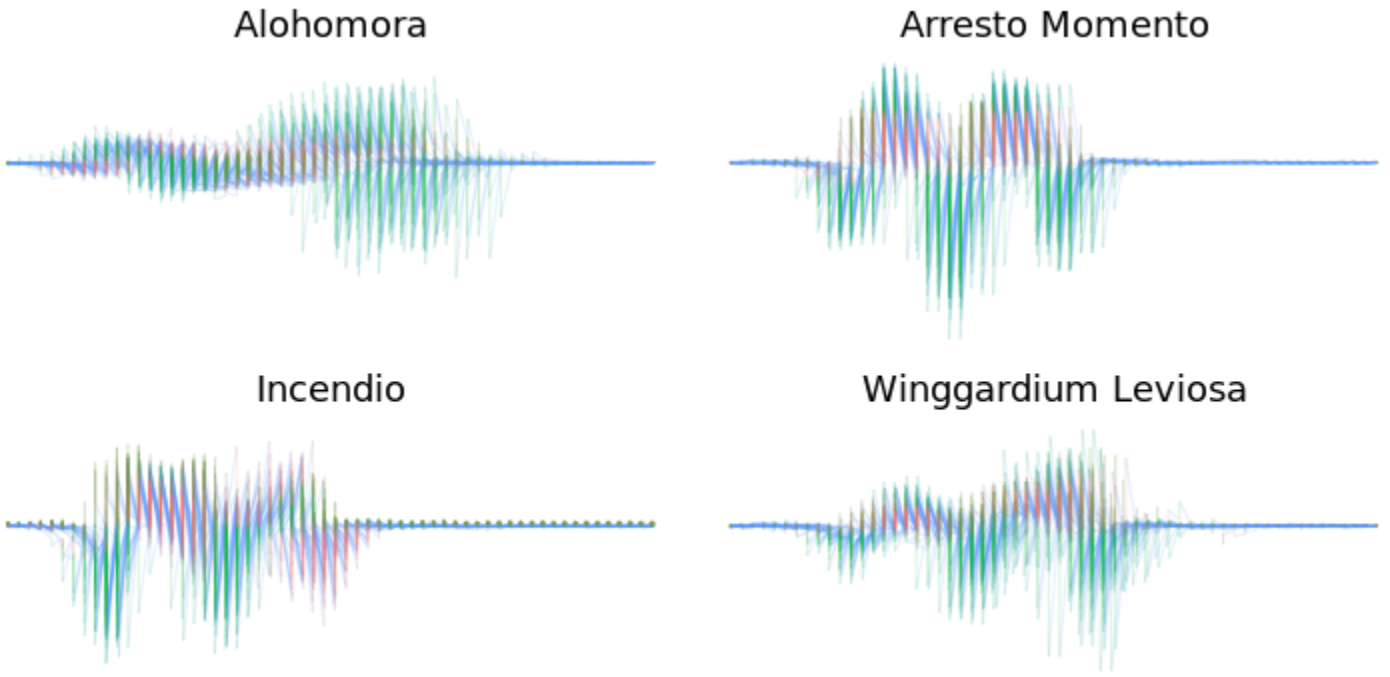
By saving all of the data to the home repo as unique id'd `.csvs` all I have to do is read them in with a single `purrr` line.

```
# get the files in our directory
all_files <- list.files('gather_data')

# just get the csvs
csv_names <- all_files[grepl('.csv', all_files)]

# load all data into a dataframe
data <- csv_names %>%
  map_df(read_csv)
```

By far the best thing about this whole project.

Visualizing all data for trends

```
ggplot(data, aes(x = time, y = accel, group = recording_id)) +
  geom_line(aes(color = direction), alpha = 0.15) +
  facet_wrap(~label) +
  theme_void() + guides(color = FALSE) +
  theme(strip.text = element_text(family = "Amatic SC", size = 18))
```



Modeling

Converting data to tensors

Currently the largest sticking point in the R world for deep learning. (*opinion*)

We (R users) like to think flat and tidy, tensors do not.

```
directions <- c('m_x','m_y','m_z')
wide_data <- data %>%
  # take data and make it wide. Et tu brutus?
  spread(direction, accel) %>%
  # group into each individual spell cast
  group_by(label, recording_id) %>%
  # convert each grouping into a list with a matrix inside.
  do( data = as.matrix(.[directions]) )
```

One-hot encoding is even more messy at this point.

Building Keras model

When building a deep learning model you need to ask yourself three things

1. What architecture fits my problem/limitations best?
2. How do I stack my layers?
3. Couldn't KNN do this better?

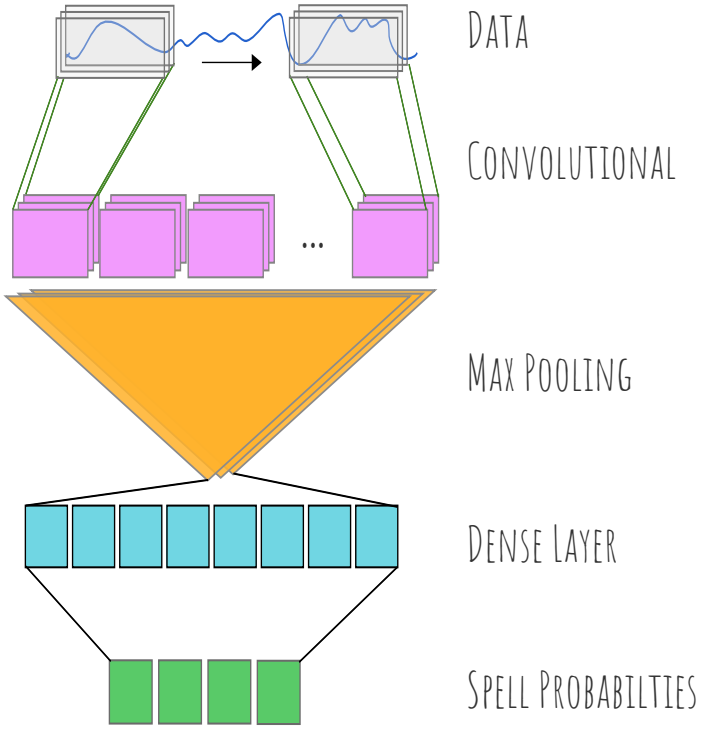
Architecture

For sequential data in deep learning there are two options, recurrent and convolutional.

Reccurent is fantastic for learning time dependent patterns but are slow.

Convolutional is great at recognizing patterns and are fast, but bad at time dependent patterns.

Since I don't have a ton of compute and accelerometer data is often pattern recognized I chose a CNN.



R's Keras vs Python's

This is where R shines in deep learning.

The pipe operator turns the sequential API into a much more readable format than in python.

R

```
model <- keras_model_sequential(
model %>%
  layer_conv_1d(...) %>%
  layer_batch_normalization() %>
  layer_global_max_pooling_1d()
  layer_dropout(0.4) %>%
  layer_dense(
    num_classes,
    activation = 'softmax'
  )
```

Python

```
model = Sequential()
model.add(conv_1d(...))
model.add(batch_normalization)
model.add(global_max_pooling_1d)
model.add(dropout(0.4))
model.add(dense(
  num_classes,
  activation = 'softmax'
))
```

Saving model for future use

You can dump the keras model as its own file to be loaded up and instantly used later.

Now

```
model %>%
  save_model_hdf5('model.h5')
```

Later

```
model <- load_model_hdf5('model.
```

While you technically could do this with any model using `saveRds`, it's nice to have it be a first-class citizen.

Final Shiny app

Wire in Keras

You simply have to plop it in like any other package.

```
library(keras)
```

Make sure to turn off GPU mode! You don't need the speed.

```
use_session_with_seed(
  42,
  disable_gpu = TRUE,
  disable_parallel_cpu = FALSE
)
```

If running on a server that you use to train this can cause fun crashes due to memory access permissions.

Testing it

Problem:

Need to test the app on a phone, but how to do that?

Solution:

Use ShinyServer hosted on your own computer!

By hosting your own server this process is way faster.

Simply save your app's files and refresh your page and it's updated!

Massively speeds up the iteration process.

Making it look pretty

The most important part

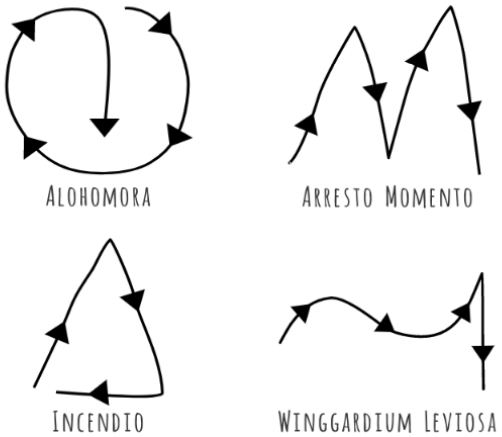
It's very easy to add some CSS to your app.

```
tags$style(HTML("
@import url('//fonts.googleapis.com/css?family=Amatic+SC|Josefin+Slab
h2, h3, button {
  font-family: 'Amatic SC', cursive;
}
h2 {
  font-size: 50px;
}
p {
  font-family: 'Josefin Slab', cursive;
  font-size: 18px;
}
button {
  font-size: 28px;
}"))
```

Before

Cast your spell!

Press the cast button and cast your spell! You have 3 seconds to perform the cast and then the model will predict which spell you performed!



Press to Cast!

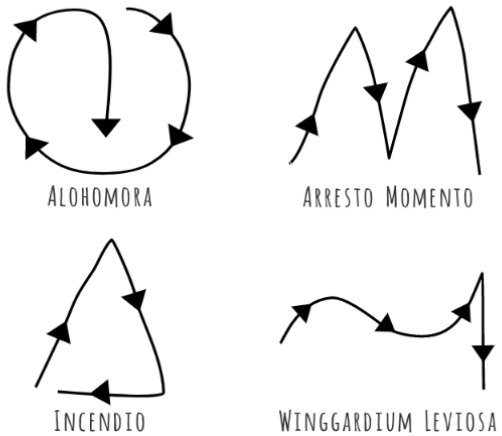
Model Predictions

Below the model will plot which spell it thinks you just cast. The higher the bar, the higher the probability the model assigns to you casting that spell.

After

Cast your spell!

Press the cast button and cast your spell! You have 3 seconds to perform the cast and then the model will predict which spell you performed!



Press to Cast!

Model Predictions

Below the model will plot which spell it thinks you just cast. The higher the bar, the

Deploying it

Do we want to kill our server?

Yes

Leave it on personal server.

No

Send it to shinyapps.io

jhubiostatistics.shinyapps.io/cast_spells/

Thanks To

- Johns Hopkins Data Science Lab
  - Support in developing `shinysense` and all the hosting time
- Lynn Bender
  - For organizing this conference
- RStudio
  - For all the amazing packages that they put out.

github repo