PCA Interpretation

2023-04-30

```
library(ggplot2)
library(knitr)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.1     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.1     v tidyr     1.3.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(pracma)
```

```
##
## Attaching package: 'pracma'
##
## The following object is masked from 'package:purrr':
##
##     cross
```

```
library(oro.nifti)
```

```
## oro.nifti 0.11.4
##
## Attaching package: 'oro.nifti'
##
## The following object is masked from 'package:pracma':
##
##     magic
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
# load(file = file.path(root_dir, "R_data", "pca_wm.Rda"))
# load(file = file.path(root_dir, "R_data", "pca_gm.Rda"))
# load(file = file.path(root_dir, "R_data", "pca_cb.Rda"))
```

PCA is a dimensionality reduction technique used to reduce the dimensionality of large datasets of image pixel intensities. The main idea is to represent the images using a smaller set of features or variables that capture most of the variance in the data. These features are the principal components, which are linear combinations of the original pixel intensities. For example, if we have values of $0.1, 0.5, 0.2$ as intensities, then $x_1 * 0.1 + x_2 * 0.5 + x_3 * 0.2$ is a linear combination of the image intensities. If we want to find out information about the image, specifically which sections show the most variance, we can look at the PCA loading vectors

The loading vectors or coefficients of the principal components indicate the contribution of each original pixel intensity to the respective principal component. A higher loading value (positive or negative) indicates a more significant contribution of the corresponding pixel intensity to the respective principal component.

To calculate the loading vectors or coefficients, we need to invert the PCA transformation by multiplying the pseudo-inverse of the original pixel intensity matrix with the principal component matrix. In this case, the A_pinv matrix is the pseudo-inverse of the original pixel intensity matrix, and the b matrix contains the principal components. The resulting x matrix contains the loading vectors or coefficients.

The singular values or PC weights (pca_gm$sdev) are multiplied by the loading vectors or coefficients to get the final normalized coefficients. The normalization step scales the coefficients to values between 0 and 1, which can be used to generate a new image that highlights the regions of the brain that contribute the most to the principal components.

The resulting image should show brighter values in the regions of the brain that are most important for distinguishing between the different classes of patients (CN, MCI, and AD) in the original dataset.

Below is in theory what it should do

```
img_dir <- 'preprocessed/imgsss'

# creating full list of gm and wm files
all_MRI_files <- get_ordered_files(file.path(root_dir, img_dir), 'smt')

# splitting into wm and gm
gm_files <- all_MRI_files[[1]]

# load all Nifti object images
nifti_images_gm <- lapply(gm_files[1:4], readNIfTI)

# these are the same steps ive done before where i grab an images and stack them as vectors
img <- c(nifti_images_gm[[1]][,,30:80])
img2 <- c(nifti_images_gm[[2]][,,30:80])
img3 <- c(nifti_images_gm[[3]][,,30:80])
img4 <- c(nifti_images_gm[[4]][,,30:80])


#### Sub proofs of concept
# data equal to suset way
sum(nifti_images_gm[[1]]@.Data[,,30:80] == nifti_images_gm[[1]][,,30:80])
```

```
## [1] 894795
```

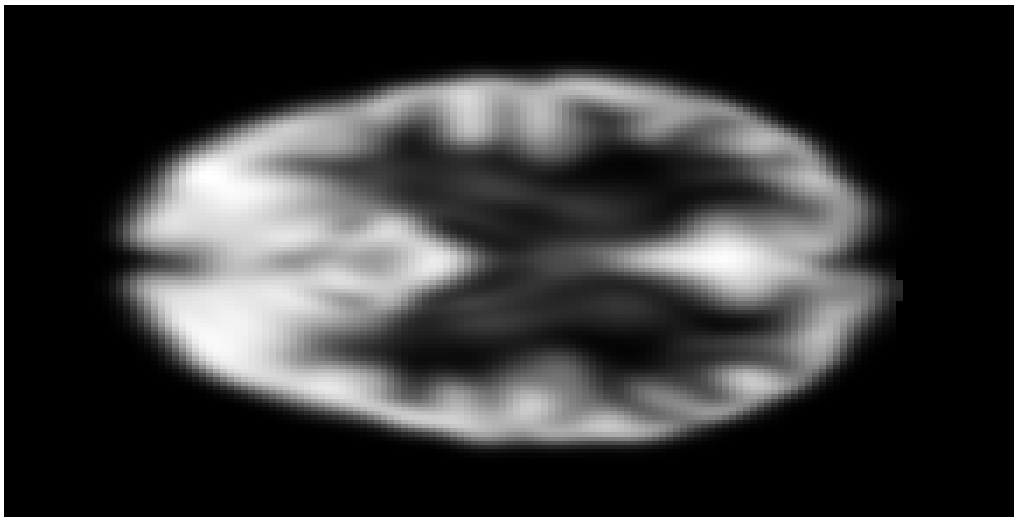
```
# original image equal to reshaped vector
sum(nifti_images_gm[[1]][,,30:80] == array(img, dim = c(121, 145, 51)))
```

```
## [1] 894795
```

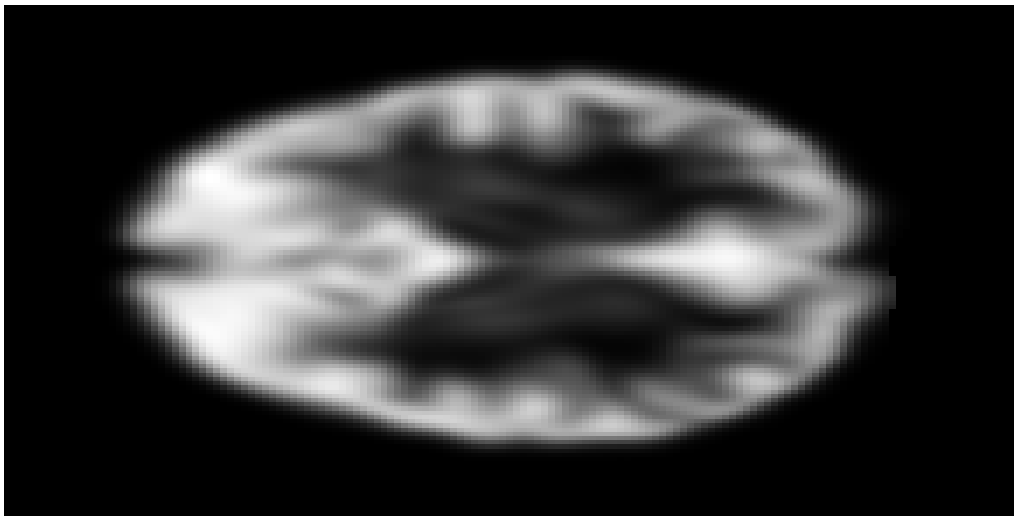
```
# stack image
test <- rbind(img, img2, img3, img4)

# remove the 0 variance columns to perform pca, not going to do here
nzv <- which(apply(test,2,var) == 0)
test_new <- test[,-nzv]

# function to repopulate the vector based off of the nzv columns with 0 values
new_img <- find_contibuting_regions(nzv, test_new[1,], slice=55)
```

```
## Warning in img_vec[-which(img_vec == 0)] <- loading_vec: number of items to
## replace is not a multiple of replacement length
```

```
#### plotting the image that was repopulated with 0s
slice <- new_img[,,35]

x <- seq(1, ncol(slice))
y <- seq(1, nrow(slice))

# plot the image intensity values
image(x, y, t(slice), col = gray(0:255/255), axes = FALSE)
```




```
#### Compare to original image
slice <- nifti_images_gm[[1]][,,65] # use slice 65 bc grabbed images 30:80

x <- seq(1, ncol(slice))
y <- seq(1, nrow(slice))

# plot the image intensity values
image(x, y, t(slice), col = gray(0:255/255), axes = FALSE)
```




```
# images are practically the same
```

```
# coef_vec_wm <- calc_loading_vec(pca_wm, n_components=160)
# save(file.path(root_dir, "R_data", "coef_vec_wm.Rda"))

# coef_vec_gm <- calc_loading_vec(pca_gm, n_components=144)
# save(file.path(root_dir, "R_data", "coef_vec_gm.Rda"))

# coef_vec_cb <- calc_loading_vec(pca_cb, n_components=157)
# save(file.path(root_dir, "R_data", "coef_vec_cb.Rda"))
```