EigenShoes

Ethan Haley

9/17/2021

```
jpgs = list.files(path = "../eigenshoe", pattern = ".jpg")
jpgs
```

```
##  [1] "RC_2500x1200_2014_us_53446.jpg" "RC_2500x1200_2014_us_53455.jpg"
##  [3] "RC_2500x1200_2014_us_53469.jpg" "RC_2500x1200_2014_us_53626.jpg"
##  [5] "RC_2500x1200_2014_us_53632.jpg" "RC_2500x1200_2014_us_53649.jpg"
##  [7] "RC_2500x1200_2014_us_53655.jpg" "RC_2500x1200_2014_us_53663.jpg"
##  [9] "RC_2500x1200_2014_us_53697.jpg" "RC_2500x1200_2014_us_54018.jpg"
## [11] "RC_2500x1200_2014_us_54067.jpg" "RC_2500x1200_2014_us_54106.jpg"
## [13] "RC_2500x1200_2014_us_54130.jpg" "RC_2500x1200_2014_us_54148.jpg"
## [15] "RC_2500x1200_2014_us_54157.jpg" "RC_2500x1200_2014_us_54165.jpg"
## [17] "RC_2500x1200_2014_us_54172.jpg"
```

```
library(jpeg)
```

Load all 17 shoe images into a matrix

Put all the pixels into one 2-D matrix – one long, flattened column for each photo.

```
T = matrix(0, 9e6, length(jpgs))
for (j in 1:length(jpgs)){
  img = readJPEG(paste("../eigenshoe/", jpgs[j], sep=''))
  T[,j] = c(as.vector(img[,,1]), as.vector(img[,,2]), as.vector(img[,,3]))
}
dim(T)
```

```
## [1] 9000000      17
```

Mean-center every pixel by row: Each of the 9M pixel positions has a mean over the 17 images, and we subtract that mean from the 17 values for each position. [This wasn't obvious from some things I read, which just said subtract the mean (of what??) from all pixels. I'm going with the wikipedia article which has MatLab code indicating `mean(T, 2)`, or row-wise mean, which makes most intuitive sense to me.]

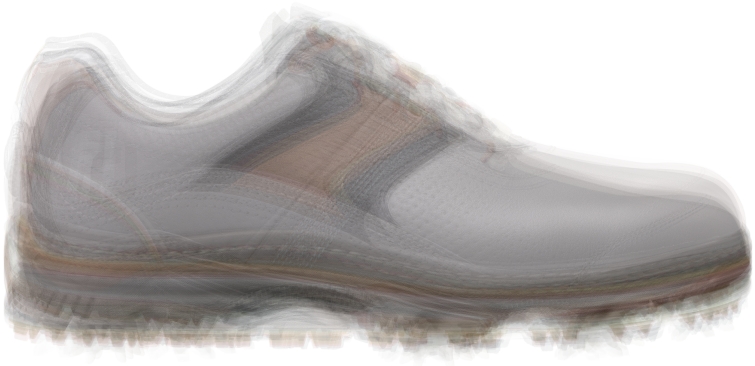
```
avgPic = rowMeans(T)
T = T - avgPic
```

Display first shoe:

```
j1 = readJPEG(paste("../eigenshoe/", jpgs[1], sep=''))
plot(1:2, type='n', xlab='', ylab='', axes=F,
     main="Shoe #1 of 17")
rasterImage(j1, 1, 1, 2, 2)
```

Display “Avg shoe”:

```
meanPic = array(avgPic, dim=c(1200,2500,3))
plot(1:2, type='n', xlab='', ylab='', axes=F,
     main="Average Shoe")
rasterImage(meanPic, 1, 1, 2, 2)
```



Display first_shoe - avg_shoe:

```
s1_mean_centered = array(pmax(T[,1], 0), dim=c(1200,2500,3))
plot(1:2, type='n', xlab='', ylab='', axes=F,
     main="First Shoe minus Avg. Shoe\n(I.e. where does this shoe differ most from any old shoe)")
rasterImage(s1_mean_centered, 1, 1, 2, 2)
```

First Shoe minus Avg. Shoe
(I.e. where does this shoe differ most from any old shoe)

The first shoe has the FJ logo near the heel, it lacks the darker/wider instep below the laces, and it has a unique texture.

Calculate the eigenshoes

If we make a covariance matrix from TT', it'll have 81 trillion values, which will break our calculations/memory.
Instead, try SVD.

```
Tsvd = svd(T)
```

First eigenshoe, min and max pixel values:

```
paste("min pixel val for first eigenshoe vector: ", min(Tsvd$u[,1]))
```

```
## [1] "min pixel val for first eigenshoe vector:  -0.00107441422273623"
```

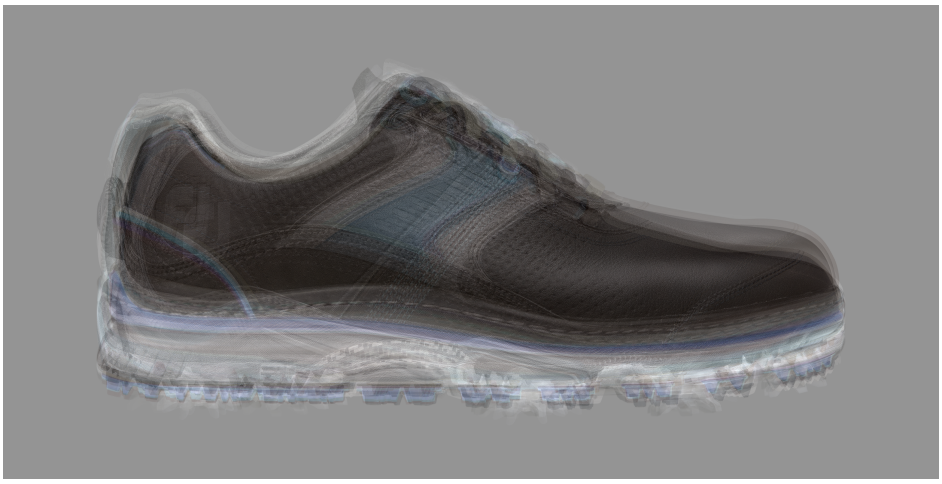
```
paste("max pixel val: ", max(Tsvd$u[,1]))
```

```
## [1] "max pixel val:  0.000778914009411712"
```

Everything will be black with that range of values, so just to be able to visualize it, divide every value by the range and then add enough to make everything be between 0 and 1

```
plotEigenshoe <- function(n, U, dim=c(1200,2500,3)) {
  u_range = max(U[,n]) - min(U[,n])
  scaled_u = U[,n] / u_range
  scaled_u = scaled_u - min(scaled_u)
  scaled_u = array(scaled_u, dim=dim)
  plot(1:2, type='n', xlab='', ylab='', axes=F,
     main=paste("Eigenshoe number", n))
rasterImage(scaled_u, 1, 1, 2, 2)
}
```

```
plotEigenshoe(1, Tsvd$u)
```



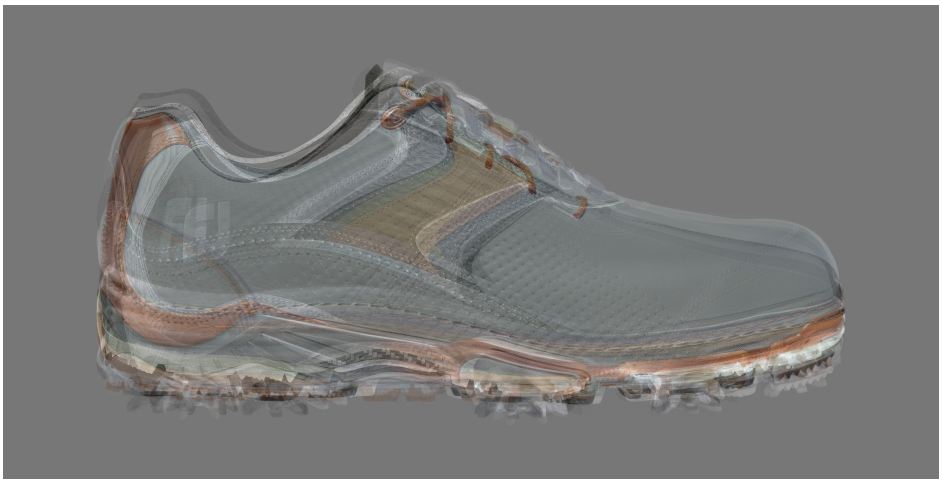
The first eigenshoe, the vector which highlights the parts of the 17 shoes that most differentiate one from another, suggests that the pixel values around the soles of the shoe describe what type of shoe each shoe is. There is also apparently something important about the color blue in that same area.

Then around the top of the shoe, it appears that the height of the shoe may give a slight indication of what type of shoe it is. Finally, the markings on the shoes below the laces (decorations/logos?) and at the heel are indicators, at least to this eigenshoe with the highest magnitude.

Let's see how this one differs from e.g. eigenshoe # 11:

```
plotEigenshoe(11, Tsvd$u)
```

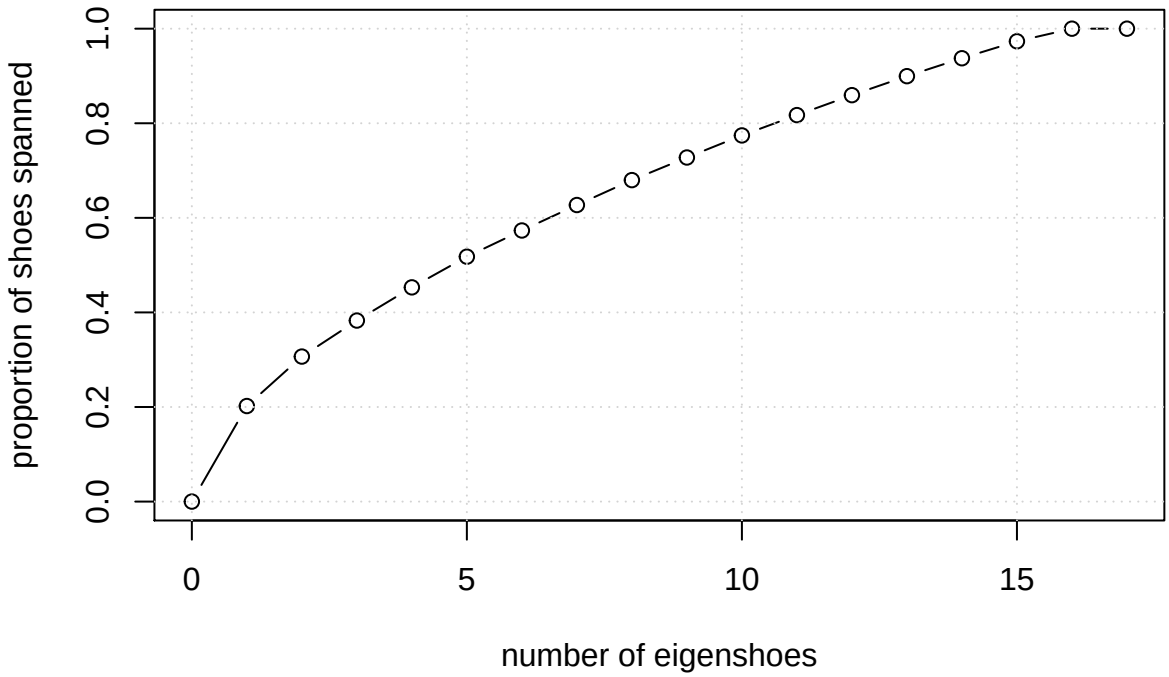
Eigenshoe number 11



That one seems to differentiate shoes based on a yellowish color. But since I had to scale these eigenshoe values to make them visible here, it's quite possible the color is different. For example, the red channel may have been overinflated along with the green and blue channels during scaling, and red values might change appearance more quickly to the eye than green and blue channels do, even if all three get scaled higher at the same rate.

How many of the 16 eigenvectors do we need, if we want to have a basis to span 80% of the variance in the 17 shoes?

```
plot(0:17, c(0, cumsum(Tsvd$d) / cumsum(Tsvd$d)[17]), type='b',
     xlab='number of eigenshoes', ylab='proportion of shoes spanned')
grid()
```



That first eigenshoe only accounted for 20% of the variance in the 17 shoes.
If our goal is to account for 80%, the plot above shows that we need to use some linear combination of at least 11 of the eigenshoes.

Here are a few more of the top 10, in case you want to take a peek.

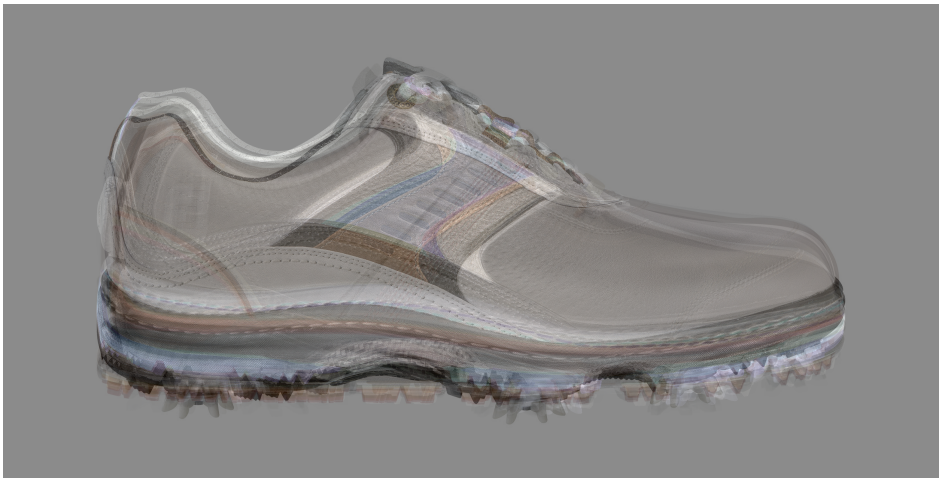
```
plotEigenshoe(2, Tsvd$u)
```


```
plotEigenshoe(3, Tsvd$u)
```

Eigenshoe number 3

```
plotEigenshoe(4, Tsvd$u)
```

Eigenshoe number 4



```
plotEigenshoe(5, Tsvd$u)
```

Eigenshoe number 5

```
plotEigenshoe(6, Tsvd$u)
```

Eigenshoe number 6