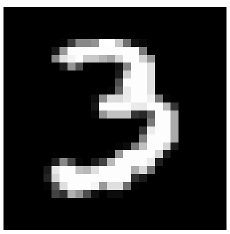
Principal Component Analysis

Load the MNIST digit recognition dataset into R from Kaggle https://www.kaggle.com/c/digit-recognizer/data

Each image is 28 by 28 pixels for a total of d = 784 pixels. Each pixel value is an integer between 0 and 255. We make use of the training dataset only, called 'train.csv', which has 785 columns, the first column being the label of the image: the true identity of the digit drawn by the user.

```
data <- read.csv("train.csv")

m <- matrix(unlist(data[10,-1]), nrow=28, byrow=TRUE)
par(pty="s")
image(t(m)[,nrow(m):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
```



```
# Consider images representing the number 3
x3 <- data[data$label==3, ]
# Remove the column of labels, and scale the pixel value between 0 and 1
x3 <- x3[,-1]/255
n <- 1000
# Take a subset of the data
x3 <- x3[seq(1,n), ]
x3 <- as.matrix(x3);
# Average pixel value - useful later
mx3 <- colMeans(x3); mx3m <- matrix(mx3, nrow=28, byrow=TRUE)
```

1. Calculating and plotting PCs

By default, `prcomp` calculates principal components on a centered version of the data.

```
pr.out <- prcomp(x3)
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

`$center` returns the column means of the original matrix, removed for principal component analysis.

```
mx3[710:715]
```

```
##   pixel709   pixel710   pixel711   pixel712   pixel713   pixel714
## 0.02653725 0.03220392 0.03644314 0.03269020 0.03095686 0.02540784
```

```
pr.out$center[710:715]
```

```
##    pixel709    pixel710    pixel711    pixel712    pixel713    pixel714
## 0.02653725 0.03220392 0.03644314 0.03269020 0.03095686 0.02540784
```

```
z <- pr.out$x
```

`$sdev` returns the square roots of the first eigenvalues of the covariance matrix. You need to square these to obtain the variance explained by each principal component.

```
pr.out$sdev[1:10]
```

```
##  [1] 2.336946 2.079064 1.941690 1.538186 1.417980 1.270983 1.167208
##  [8] 1.115882 1.111711 1.029678
```

```
pr.var=pr.out$sdev^2
pve=pr.var/sum(pr.var)   # Proportion of variance explained
pve[1:10]
```

```
##  [1] 0.12144380 0.09611999 0.08383736 0.05261332 0.04471141 0.03592176
##  [7] 0.03029525 0.02768944 0.02748285 0.02357659
```

The rotated data is extracted using `$rotation`

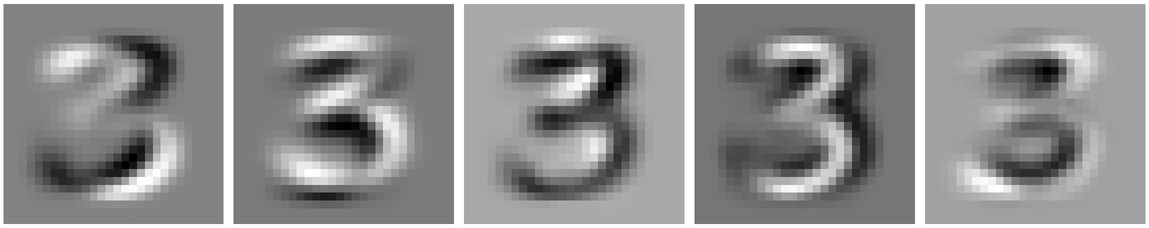
```
# Rotation matrix whose columns contain the eigenvectors = principal components
R <- pr.out$rotation
dim(R)
```

```
## [1] 784 784
```

The original data is of dimension $d = 784$, and the sample size is $n = 1000 > d:$ we end up with 784 principal components. Start again with a smaller subset, say $n = 100$ data points. What is the dimension of R? Why?

Plotting the first 5 principal components:

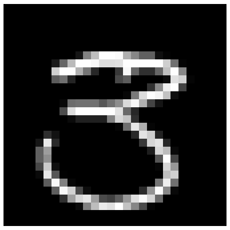
```
pc1 <- R[,1]; pc1m <- matrix(pc1, nrow=28, byrow=TRUE)
pc2 <- R[,2]; pc2m <- matrix(pc2, nrow=28, byrow=TRUE)
pc3 <- R[,3]; pc3m <- matrix(pc3, nrow=28, byrow=TRUE)
pc4 <- R[,4]; pc4m <- matrix(pc4, nrow=28, byrow=TRUE)
pc5 <- R[,5]; pc5m <- matrix(pc5, nrow=28, byrow=TRUE)
par(mfrow=c(1,5), mar=c(.2,.2,.2,.2), pty="s")
image(t(pc1m)[,nrow(pc1m):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(pc2m)[,nrow(pc2m):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(pc3m)[,nrow(pc3m):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(pc4m)[,nrow(pc4m):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(pc5m)[,nrow(pc5m):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
```



2. Image reconstruction from PCs

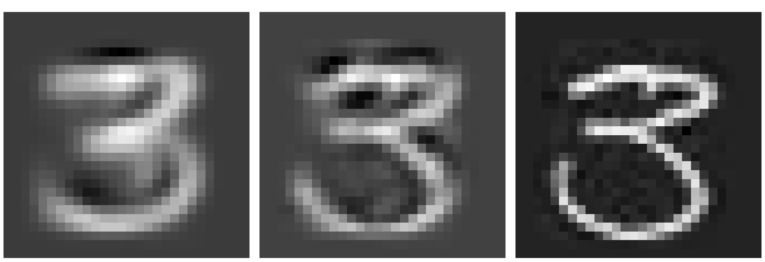
Consider the first image:

```
par(pty="s")
x31 = matrix(x3[1, ], nrow=28, byrow=TRUE)
image(t(x31)[,nrow(x31):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
```



We approximate the original image keeping 5, 50 and 200 principal components. What we are effecively doing in `reconstruction5` is calculating `z[1,1]*pc1 + z[1,2]*pc2 + z[1,3]*pc3 + z[1,4]*pc4 + z[1,5]*pc5 + mx3`.

```
reconstruction5   <- rowSums(R[, 1:5]%*%diag(z[1, 1:5])) + mx3
reconstruction50  <- rowSums(R[, 1:50]%*%diag(z[1, 1:50])) + mx3
reconstruction200 <- rowSums(R[, 1:200]%*%diag(z[1, 1:200])) + mx3
par(mfrow=c(1,3), mar=c(.2,.2,.2,.2), pty="s")
rec5 = matrix(reconstruction5, nrow=28, byrow=TRUE)
rec50 = matrix(reconstruction50, nrow=28, byrow=TRUE)
rec200 = matrix(reconstruction200, nrow=28, byrow=TRUE)
image(t(rec5)[,nrow(rec5):1], axes = FALSE,
       col = grey(seq(0, 1, length = 256)))
image(t(rec50)[,nrow(rec50):1], axes = FALSE,
       col = grey(seq(0, 1, length = 256)))
image(t(rec200)[,nrow(rec200):1], axes = FALSE,
       col = grey(seq(0, 1, length = 256)))
```



3. Using your knowledge of SVD decomposition

The principal components of X (n by d matrix) correspond equivalently to

(i) The eigenvectors of the sample covariance matrix $S = (X^t X)/n$

(ii) The columns of V in the SVD decomposition $X = ULV^t$ of the observation matrix

In addition, if l denotes a singular value of X, then l^2/n is en eigenvalue of the sample covariance matrix S.

```r
x3c <- scale(x3, center = TRUE, scale = FALSE)  # Centre the observation matrix
x.svd <- svd(x3c)           # SVD decomposition
L <- x.svd$d                # Singular values of X
U <- x.svd$u                # Matrix of left singular vectors
V <- x.svd$v                # Matrix of right singular vectors = PCs
```

Compare the value `z <- pr.out$x` returned by `prcomp`, with UL and XV; they are the same.

```r
x.pc1 <- U%*%diag(L)
x.pc2 <- x3c%*%V
# First 5 values for the first image
z[1,1:5]
```

```
##         PC1         PC2         PC3         PC4         PC5
##  0.91048740 -2.71449852  0.02745269 -2.45597708  2.15914167
```

```r
x.pc1[1,1:5]
```

```
## [1]  0.91048740 -2.71449852  0.02745269 -2.45597708  2.15914167
```

```r
x.pc2[1,1:5]
```

```
## [1]  0.91048740 -2.71449852  0.02745269 -2.45597708  2.15914167
```

Next consider the eigenvalue-eigenvector decomposition of the sample covariance matrix S.

```r
S <- t(x3c)%*%x3c/n         # Sample correlation matrix
S.ee <- eigen(S)
S.val <- S.ee$values        # Eigenvalues of S
S.vect <- S.ee$vectors      # Eigenvectors of S
```

Compare the eigenvalues of S with l^2/n, where l is a singular value of X. These are the same.

```r
S.val[1:10]                 # Compare the eigenvalues of S with ...
```

```
##  [1] 5.455856 4.318185 3.766389 2.363650 2.008658 1.613783 1.361012
##  [8] 1.243947 1.234665 1.059177
```

```r
L[1:10]^2/n                 # ... l^2/n, where l is a singular value of X
```

```
##  [1] 5.455856 4.318185 3.766389 2.363650 2.008658 1.613783 1.361012
##  [8] 1.243947 1.234665 1.059177
```

Compare these with the eigenvalues returned by the function prcomp: he slight discrepancy is due to the factor $n-1$ used by `prcomp` instead of n.

```
pr.var[1:10]
```

```
##  [1] 5.461317 4.322507 3.770159 2.366016 2.010668 1.615398 1.362375
##  [8] 1.245192 1.235901 1.060237
```

```
L[1:10]^2/(n-1)
```

```
##  [1] 5.461317 4.322507 3.770159 2.366016 2.010668 1.615398 1.362375
##  [8] 1.245192 1.235901 1.060237
```

Next plot the first 5 principal components -> Compare with images produced in Section 1.

```
par(mfrow=c(1,5), mar=c(.2,.2,.2,.2), pty="s")
V1 = matrix(V[,1], nrow=28, byrow=TRUE)
V2 = matrix(V[,2], nrow=28, byrow=TRUE)
V3 = matrix(V[,3], nrow=28, byrow=TRUE)
V4 = matrix(V[,4], nrow=28, byrow=TRUE)
V5 = matrix(V[,5], nrow=28, byrow=TRUE)
image(t(V1)[,nrow(V1):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(V2)[,nrow(V2):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(V3)[,nrow(V3):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(V4)[,nrow(V4):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(V5)[,nrow(V5):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
```

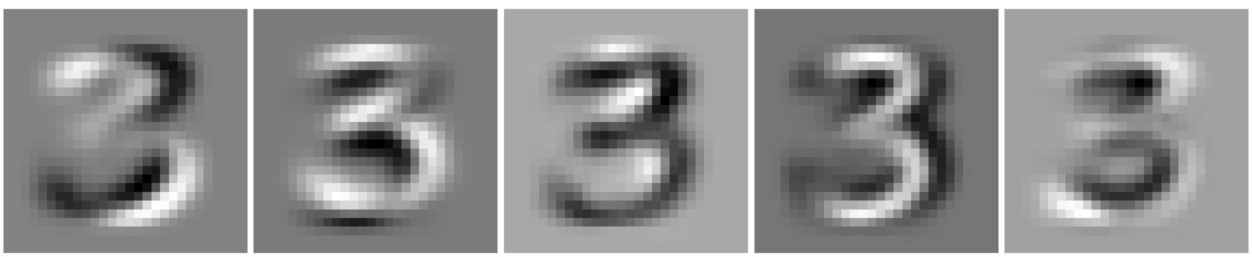
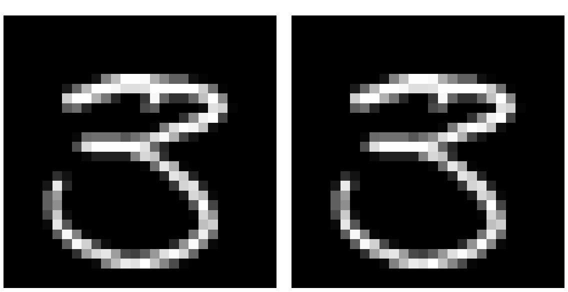


Image reconstruction -> Compare with images produced in Section 2.

```
# We recover the original image - do not forget to add the mean back:
xid <- U%*%diag(L)%*%t(V)
par(mfrow=c(1,2), mar=c(.2,.2,.2,.2), pty="s")
m.xid <- matrix(xid[1,], nrow=28, byrow=TRUE) + mx3m
image(t(x31)[,nrow(x31):1], axes = FALSE,
      col = grey(seq(0, 1, length = 256)))
image(t(m.xid)[,nrow(m.xid):1], axes = FALSE,
      col = grey(seq(0, 1, length = 256)))
```



We approximate the original image keeping 5, 50 and 200 principal components.

```
L5 <- rep(0, length(L)); L5[1:5] <- L[1:5]
L50 <- rep(0, length(L)); L50[1:50] <- L[1:50]
L200 <- rep(0, length(L)); L200[1:200] <- L[1:200]
reconstruct5 <- U%*%diag(L5)%*%t(V)
reconstruct50 <- U%*%diag(L50)%*%t(V)
reconstruct200 <- U%*%diag(L200)%*%t(V)
rec5 <- matrix(reconstruct5[1,], nrow=28, byrow=TRUE) + mx3m
rec50 <- matrix(reconstruct50[1,], nrow=28, byrow=TRUE) + mx3m
rec200 <- matrix(reconstruct200[1,], nrow=28, byrow=TRUE) + mx3m
par(mfrow=c(1,3), mar=c(.2,.2,.2,.2), pty="s")
image(t(rec5)[,nrow(rec5):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(rec50)[,nrow(rec50):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
image(t(rec200)[,nrow(rec200):1], axes = FALSE, col = grey(seq(0, 1, length = 256)))
```