PCA, SVD and Mahalanobis distance

Christopher Gillies

11/16/2017

Multivariate normal distribution

$$f(x) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)\right) \tag{1}$$

If we assume the data are zero-centered then:

$$f(x) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(x)^T \mathbf{\Sigma}^{-1}(x)\right) \tag{2}$$

Generate a random sample

```
require(MASS)
```

```
## Loading required package: MASS
```

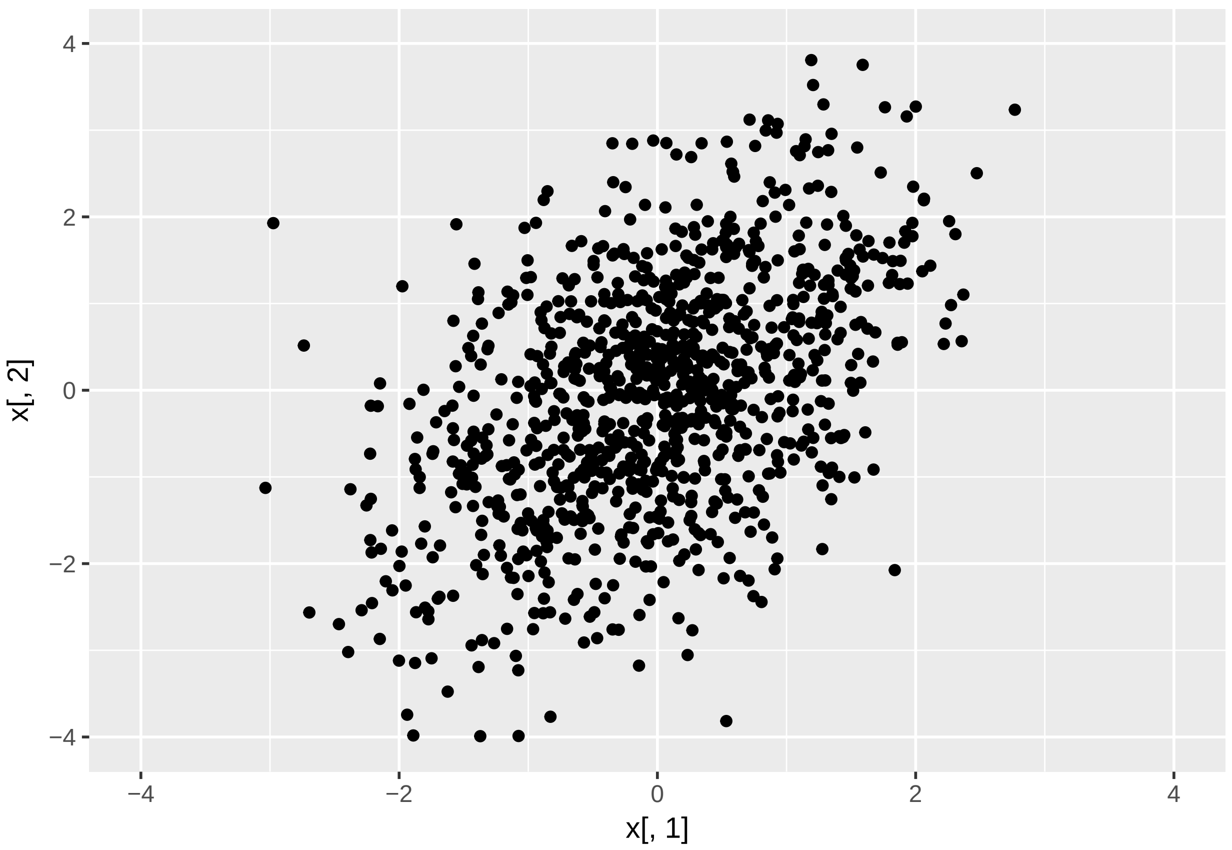
```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```
S = matrix(c(1,0.75,0.75,2),ncol=2)
x = mvrnorm(n = 1000, mu=c(0,0), Sigma=S)

ggplot(data.frame()) + geom_point(aes(x=x[,1],y=x[,2])) + scale_x_continuous(limits=c(-4,4))  + scale_y_
```

```
## Warning: Removed 7 rows containing missing values (geom_point).
```



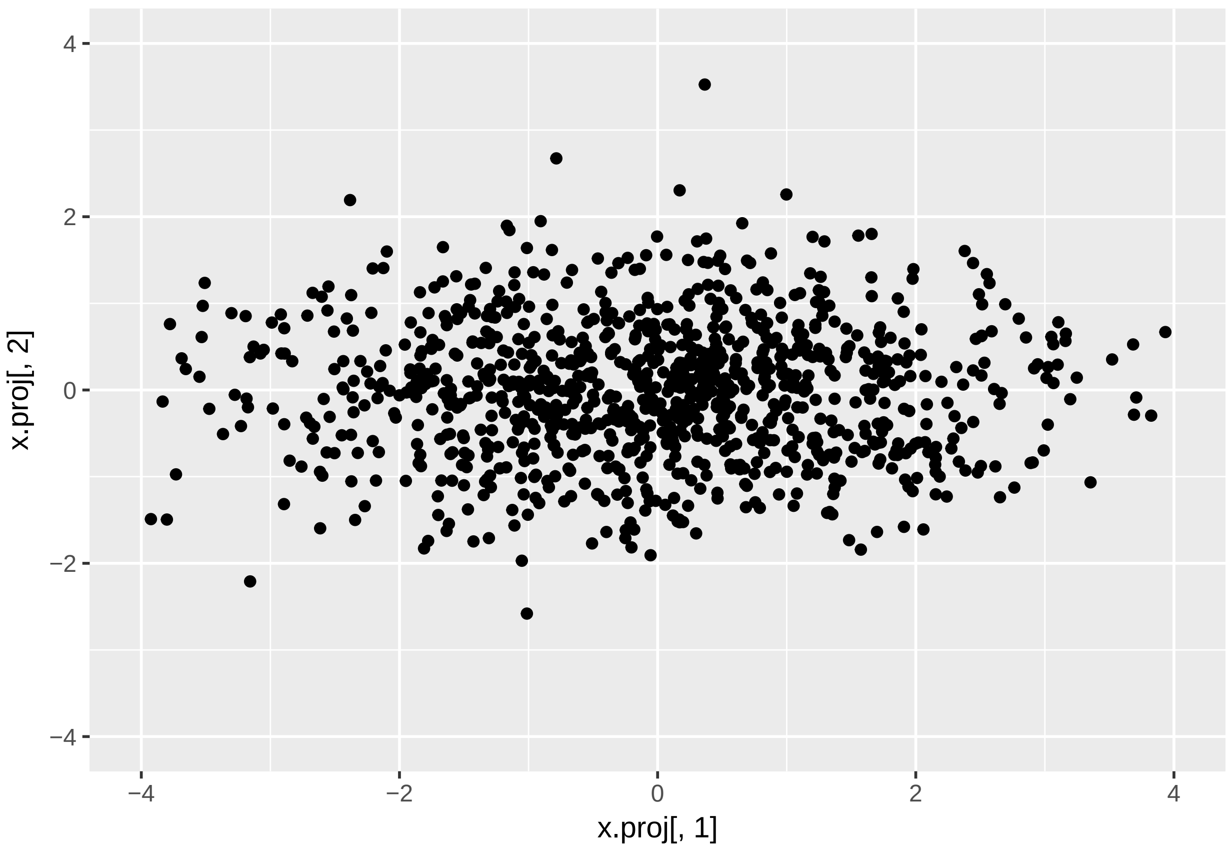
Perform PCA

```
S.sample = cov(x)

e.decomp = eigen(S.sample)

e.decomp$vectors %*% diag(e.decomp$values) %*% t(e.decomp$vectors)
```

```
##           [,1]      [,2]
## [1,] 0.9746186 0.6984203
## [2,] 0.6984203 1.9873731
```

```
x.proj = x %*% e.decomp$vectors
ggplot(data.frame()) + geom_point(aes(x=x.proj[,1],y=x.proj[,2])) + scale_x_continuous(limits=c(-4,4))
```

```
## Warning: Removed 11 rows containing missing values (geom_point).
```



```
e.decomp$values
```

```
## [1] 2.3436713 0.6183205
```

```
var(x.proj[,1])
```

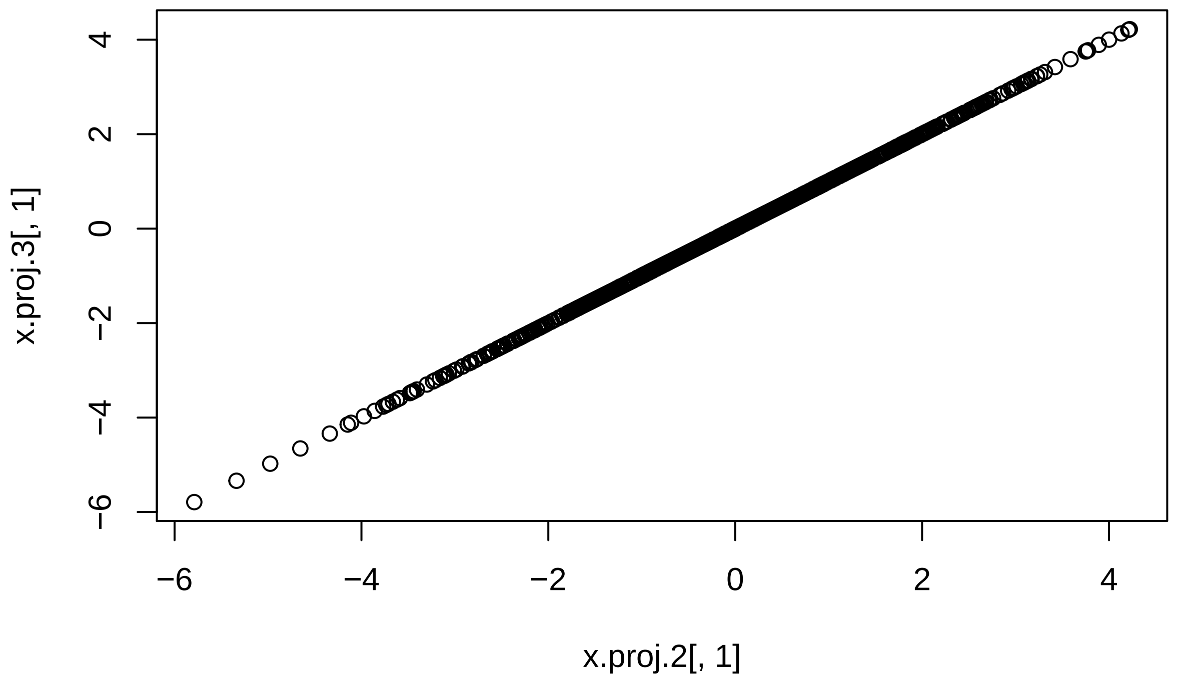
```
## [1] 2.343671
```

```
var(x.proj[,2])
```

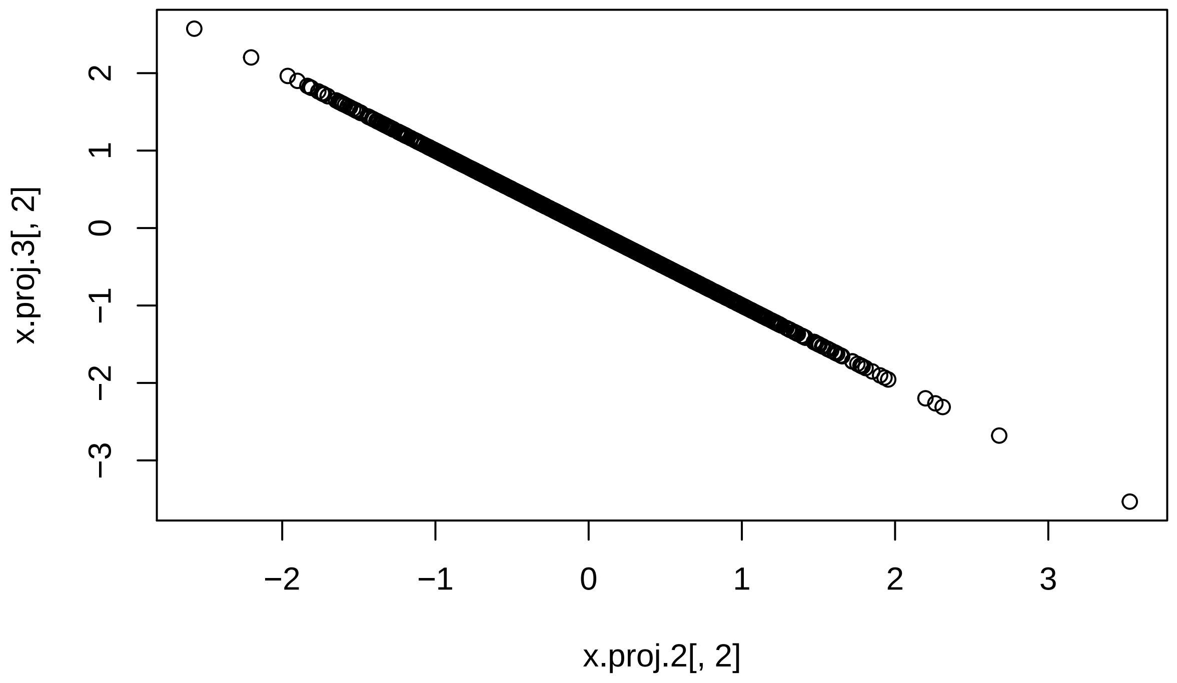
```
## [1] 0.6183205
```

Notice that the variance of the projected data matches the eigenvalues of the covariance matrix. The projection of x onto its principal components simply rotates the data.

```
svd.sample.cov = svd(S.sample)


x.center = scale(x,scale = FALSE)

svd.x = svd(x.center)


x.proj.2 = x.center %*% svd.x$v
x.proj.3 = x.center %*% svd.sample.cov$v

plot(x.proj.2[,1],x.proj.3[,1])
```



```r
plot(x.proj.2[,2],x.proj.3[,2])
```



```r
svd.x$d^2 / (1000 - 1)
```

```
## [1] 2.3436713 0.6183205
```

```r
svd.sample.cov$d
```

```
## [1] 2.3436713 0.6183205
```

The same eigenvectors and eigenvalues are computed from the matrix x.center and the covariance matrix. The singlar values of x and the cov(x) are related as follows:

$$\lambda_i = \frac{\sigma_i^2}{n-1} \tag{3}$$

$$\text{COV}\left[X\right] = \frac{1}{n-1} X^T X \tag{4}$$

$$X = U \Sigma V^T \tag{5}$$

$$X^T X = (U \Sigma V^T)^T (U \Sigma V^T) \tag{6}$$

$$X^T X = V \Sigma U^T U \Sigma V^T \tag{7}$$

$$X^T X = V \Sigma \Sigma V^T = V \Sigma^2 V^T \tag{8}$$

$$\frac{1}{n-1} X^T X = \frac{1}{n-1} V \Sigma^2 V^T \rightarrow \frac{1}{n-1} \Sigma^2 = \Lambda \tag{9}$$

where Λ is the diagonal matrix of eigenvalues of $\text{COV}\left[X\right]$.

This is the same formula as above for the relationship between the singular values of X and the eigenvalues of its covariance matrix.

What happens if we scale X before running PCA?

```
x.scaled = scale(x)

cov.scaled.x = cov(x.scaled)
cov.scaled.x
```

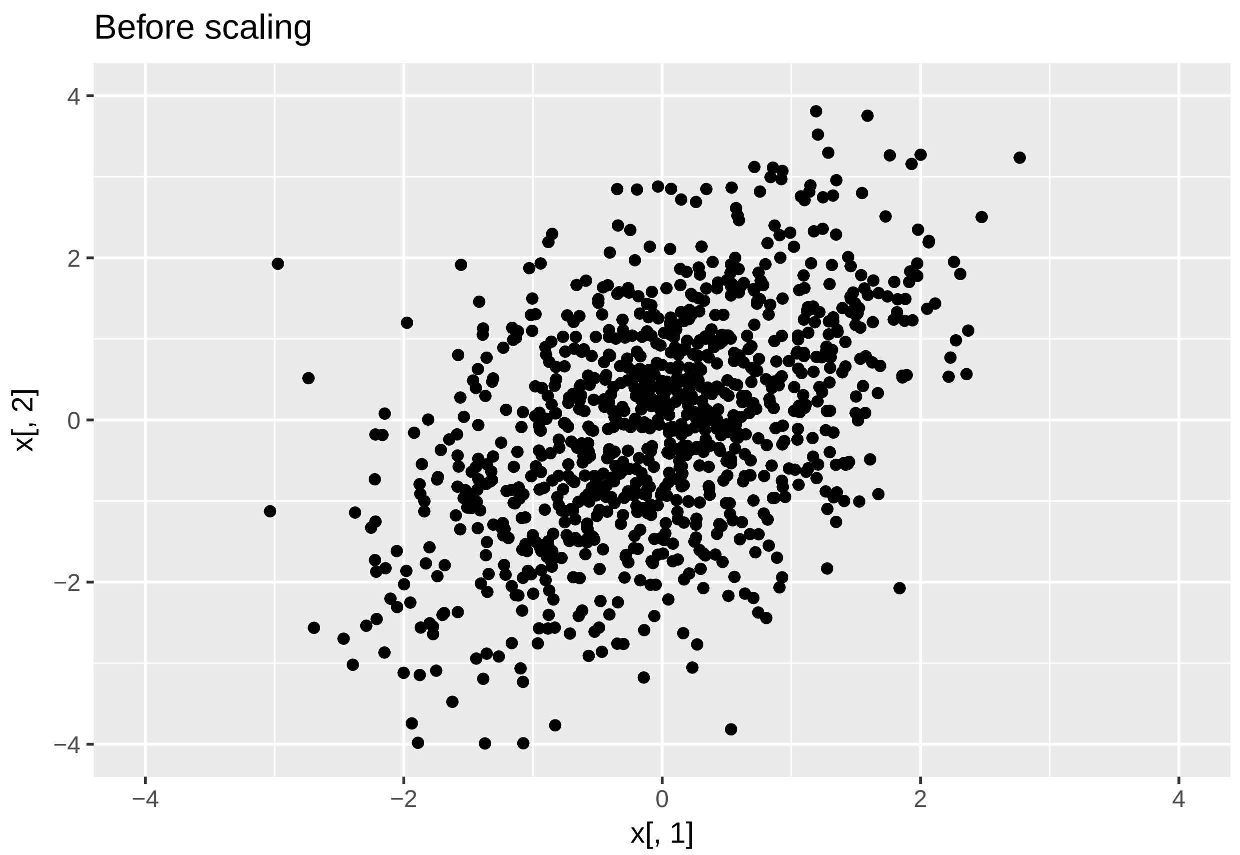
```
##           [,1]      [,2]
## [1,] 1.0000000 0.5018337
## [2,] 0.5018337 1.0000000
```

```
S.sample
```

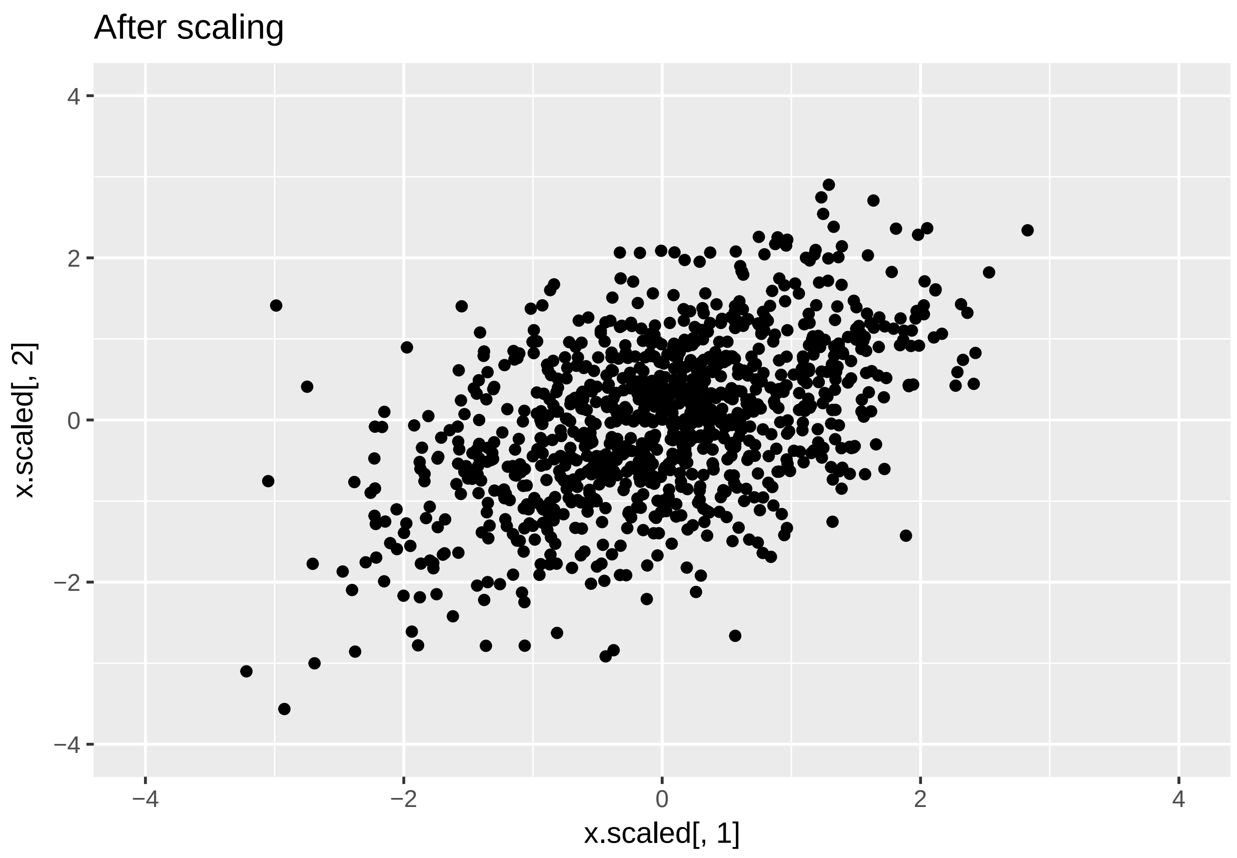
```
##           [,1]      [,2]
## [1,] 0.9746186 0.6984203
## [2,] 0.6984203 1.9873731
```

```
ggplot(data.frame()) + geom_point(aes(x=x[,1],y=x[,2])) + scale_x_continuous(limits=c(-4,4))  + scale_y
```

```
## Warning: Removed 7 rows containing missing values (geom_point).
```

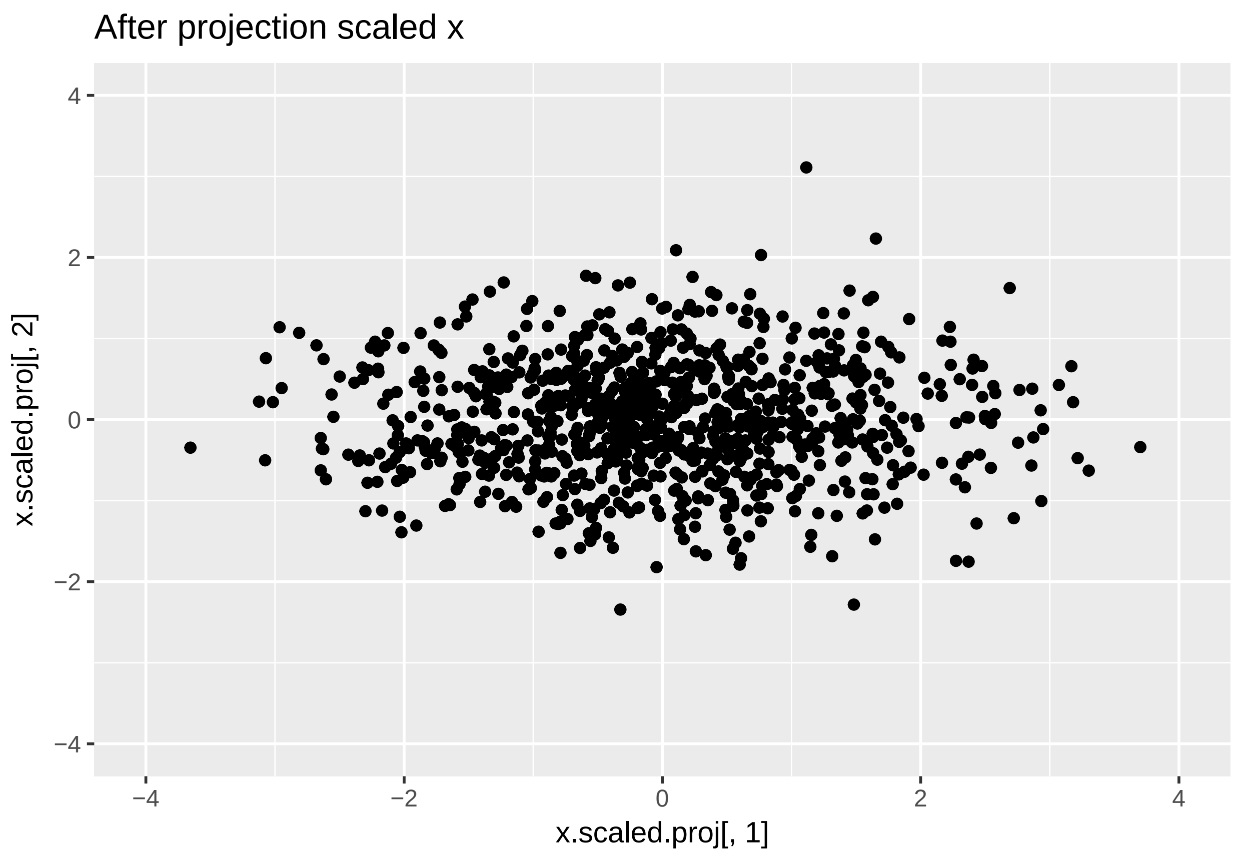


```
ggplot(data.frame()) + geom_point(aes(x=x.scaled[,1],y=x.scaled[,2])) + scale_x_continuous(limits=c(-4,4
```



```
scaled.svd = svd(cov.scaled.x)

x.scaled.proj = x.scaled %*% scaled.svd$v
ggplot(data.frame()) + geom_point(aes(x=x.scaled.proj[,1],y=x.scaled.proj[,2])) + scale_x_continuous(li
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```



```
svd.sample.cov$d
```

```
## [1] 2.3436713 0.6183205
```

```
scaled.svd$d
```

```
## [1] 1.5018337 0.4981663
```

Data reconstruction

```
x.projection = svd.sample.cov$v %*% t(x.center)
x.reconstruction = t(svd.sample.cov$v %*% x.projection)
x.reconstruction.partial = t(svd.sample.cov$v %*% rbind(x.projection[1,], 0))

cor(x.reconstruction[,1],x.center[,1])
```

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

```
cor(x.reconstruction[,2],x.center[,2])
```

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

```
cor(x.reconstruction[,1],x.reconstruction.partial[,1])
```

```
## [1] 0.7046916
```

```
cor(x.reconstruction[,2],x.reconstruction.partial[,2])
```

```
## [1] 0.9673419
```

Notince that the reconstruction works perfectly, and the partial reconstruction works fairly well.

Whitening transform

```
x.proj.white = x.center %*% svd.sample.cov$v %*% solve(diag(sqrt(svd.sample.cov$d)))
var(x.proj.white[,1])
```

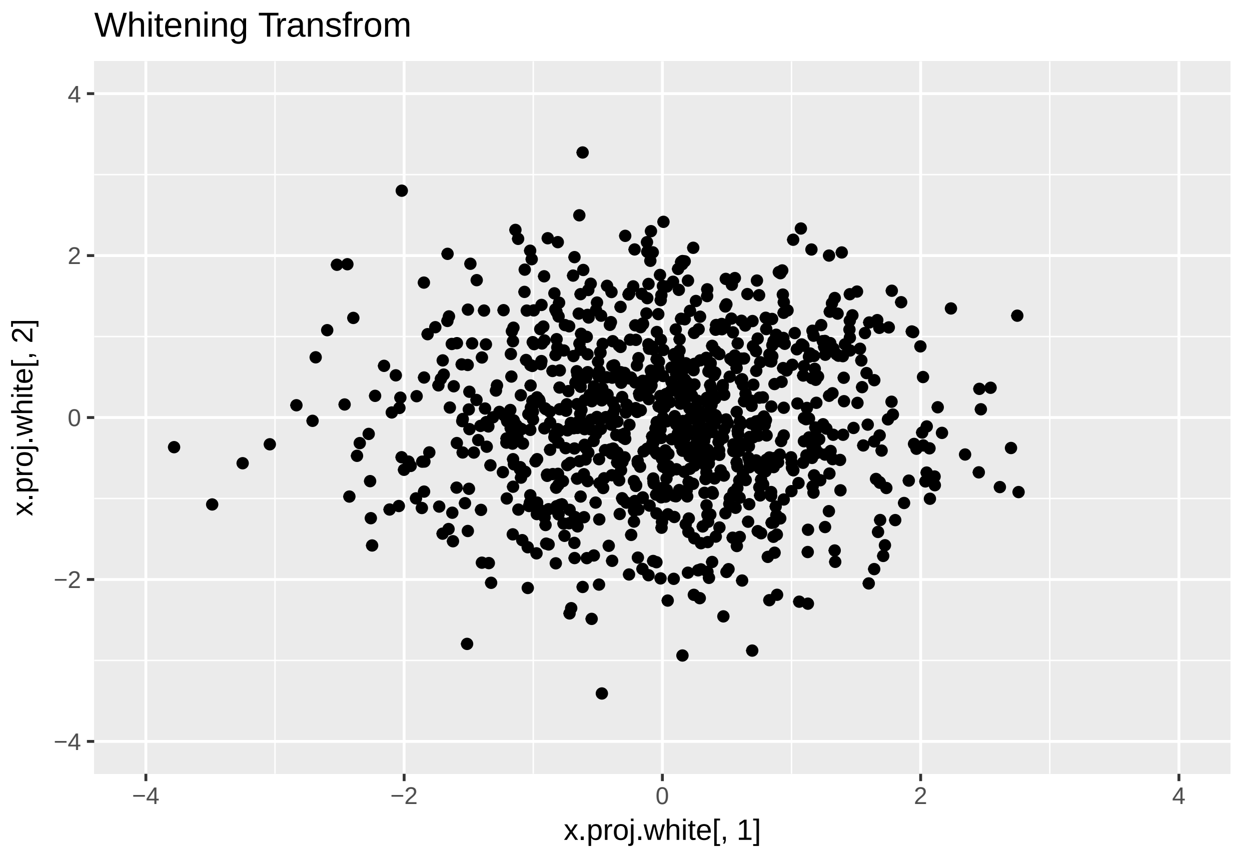
```
## [1] 1
```

```
var(x.proj.white[,2])
```

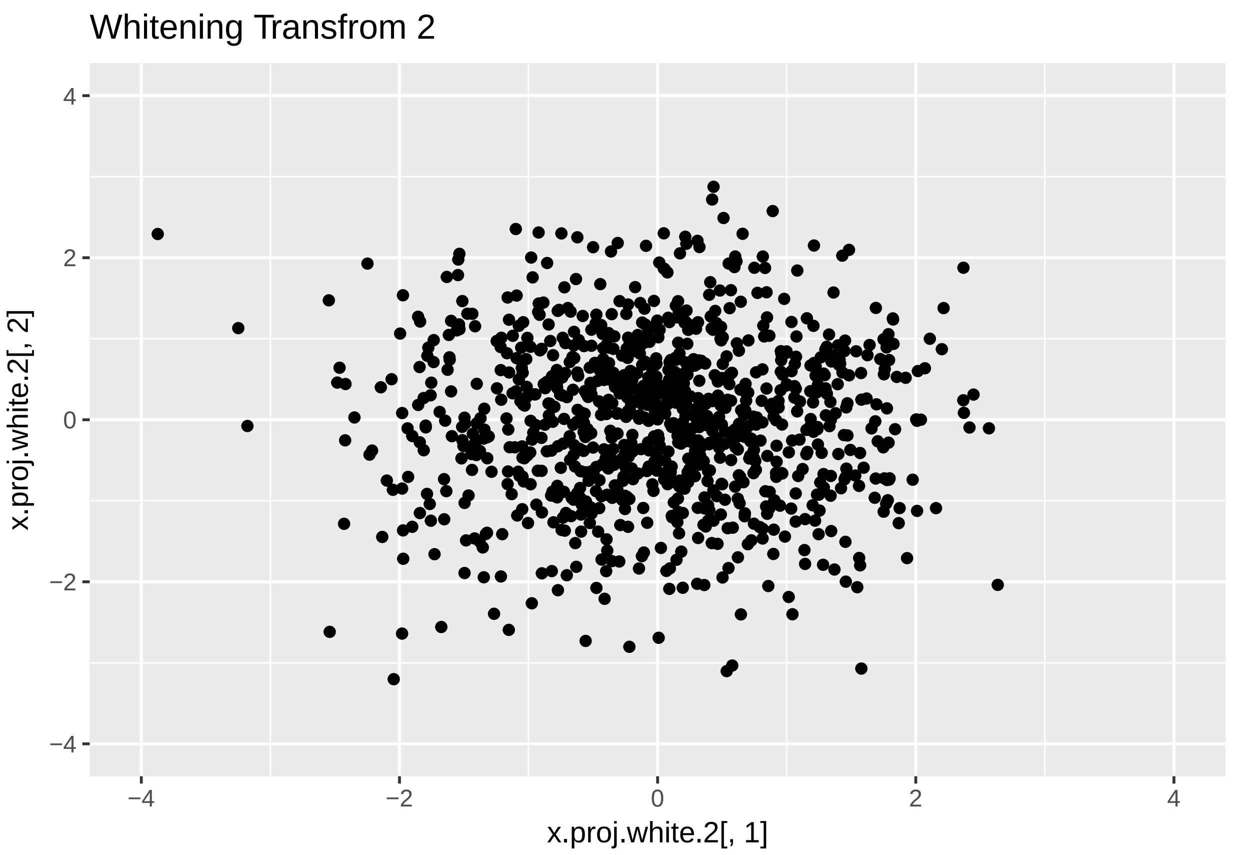
```
## [1] 1
```

```
ggplot(data.frame()) + geom_point(aes(x=x.proj.white[,1],y=x.proj.white[,2])) + scale_x_continuous(limi
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



```
w_sqrt = svd.sample.cov$v %*% diag(sqrt(svd.sample.cov$d)) %*% t(svd.sample.cov$v)
w_inv_sqrt = solve(w_sqrt)
w_inv_sqrt.2 = t(svd.sample.cov$v) %*% solve(diag(sqrt(svd.sample.cov$d))) %*% svd.sample.cov$v
x.proj.white.2= x.center %*% w_inv_sqrt.2
ggplot(data.frame()) + geom_point(aes(x=x.proj.white.2[,1],y=x.proj.white.2[,2])) + scale_x_continuous(
```



```
cov(x.proj.white.2)
```

```
##                  [,1]          [,2]
## [1,]  1.000000e+00 -1.415532e-16
## [2,] -1.415532e-16  1.000000e+00
```

```
cov(x.proj.white)
```

```
##                [,1]         [,2]
## [1,] 1.000000e+00 2.875605e-17
## [2,] 2.875605e-17 1.000000e+00
```

The above are two different whitening transforms. Please note that:

$$\mathrm{COV}[X] = V\Lambda V^T \tag{10}$$

In a diagonal matrix Λ we can take the square roots by just taking the square root of the diagonal elements

$$\mathrm{COV}[X] = (V\Lambda^{1/2}V^T)^2 = V\Lambda^{1/2}(V^TV)\Lambda^{1/2}V^T = V\Lambda V^T \tag{11}$$

Since, $V^TV = I$. That is $V^T = V^{-1}$.

$$\mathrm{COV}[X]^{1/2} = V\Lambda^{1/2}V^T \tag{12}$$

Now $(ABC)^-1 = C^{-1}B^{-1}A^{-1}$. Therefore

$$\mathrm{COV}[X]^{-1/2} = (V\Lambda^{1/2}V^T)^{-1} = V^{-1}\Lambda^{-1/2}(V^{-1})^-1 = V^T\Lambda^{-1/2}V \tag{13}$$

Since $V^T = V^{-1}$.

Mahalanobis distance^2

The Mahalanobis of a centered vector from the origin is:

$$D_M(x)^2 = x^T S^{-1} x \tag{14}$$

where S is the covariance matrix of the vector x.

```
t(x.center[1,]) %*% solve(S.sample) %*% x.center[1,]
```

```
##          [,1]
## [1,] 1.181852
```

```
sum(x.proj.white[1,]^2)
```

```
## [1] 1.181852
```

```
sum(x.proj.white.2[1,]^2)
```

```
## [1] 1.181852
```

```
x.1.proj.scaled = x.center[1,] %*% svd.sample.cov$v %*% solve(diag(sqrt(svd.sample.cov$d)))
sum(x.1.proj.scaled^2)
```

```
## [1] 1.181852
```

Mahalanobis distance^2 is the equal to the $\|x\|^2$ in the Whitened space.

The norm of a vector

$$\|x\|^2 = \sum x_i^2 = x^T x \tag{15}$$

$$(x^T)^T = x \tag{16}$$

$$D_M(x)^2 = x^T \mathrm{COV}[X]^{-1} x \tag{17}$$

$$\mathrm{COV}[X]^{-1} = (V \Lambda V^T)^{-1} = V^T \Lambda^{-1} V \tag{18}$$

The Mahalanobis distance^2 can be written as

$$D_M(x)^2 = x^T V^T \Lambda^{-1} V x = x^T V^T \Lambda^{-1/2} \Lambda^{-1/2} V x = (x^T V^T \Lambda^{-1/2})(\Lambda^{-1/2} V x) = (\Lambda^{-1/2} V x)^T (\Lambda^{-1/2} V x) \tag{19}$$

$$= (\Lambda^{-1/2} V x)^T (\Lambda^{-1/2} V x) = \|\Lambda^{-1/2} V x\|^2 = \|x^T V^T \Lambda^{-1/2}\|^2 = D_M(x)^2 \tag{20}$$

The final equality $\|x^T V^T \Lambda^{-1/2}\|^2$ shows that the $D_M(x)^2$ is the same as projecting the x onto its principal components, then scaling each axis by the square root of its eigenvalue (if the eigenvalue is the variance then the sqrt(eigenvalue) is like the standard deviation) and finally taking the norm of the scaled projected x. A eigenvalue is the variance of the data projected onto its corresponding eigenvector, so to scale it you divide by the standard deviation.

What about PCA and SVD in the case where there are more variables than observations?