Kernel PCA

Carlo Morales

The following kernels are available in `kernlab`.

- **rbfdot(sigma = 1)**
  - $k(x, y) = \exp(-\sigma\|x - y\|^2)$
- **polydot(degree = 1, scale = 1, offset = 1)**
  - $k(x, y) = (\alpha x^T y + c)^d$
- **tanhdot(scale = 1, offset = 1)**
  - $k(x, y) = \tanh(\alpha x^T y + c)$
- **vanilladot()**
  - $k(x, y) = x^T y + c$
- **laplacedot(sigma = 1)**
  - $k(x, y) = \exp(\sigma\|x - y\|)$
- **besseldot(sigma = 1, order = 1, degree = 1)**
  - $k(x, y) = -Bessel^n_{(\nu+1)}(\sigma\|x - y\|^2)$
- **anovadot(sigma = 1, degree = 1)**
  - $k(x, y) = \sum_{i=1}^n \exp(-\sigma(x_i - y_i)^2)^d$
- **splinedot()**
  - $k(x, y) = \prod_{i=1}^d \left(1 + x_i y_i + x_i y_i \cdot \min(x_i, y_i) - \frac{x_i + y_i}{2} \cdot \min(x_i, y_i)^2 + \frac{1}{3} \cdot \min(x_i, y_i)^3\right)$
  - d is the dimension of the data set being analyzed.
- **string kernel**
  - The string kernel is used with string data.

Arguments

- **sigma** - The inverse kernel width used by the Gaussian the Laplacian, the Bessel and the ANOVA kernel.
- **degree** - The degree of the polynomial, Bessel or ANOVA kernel function. This has to be an positive integer.
- **scale** - The scaling parameter of the polynomial and tangent kernel is a convenient way of normalizing patterns without the need to modify the data itself.
- **offset** - The offset used in a polynomial or hyperbolic tangent kernel
- **order** - The order of the Bessel function to be used as a kernel

How does kernel PCA work?

- Assume the data matrix X we are analyzing has size $n \times p$.
- Pick a kernel function, $k(x, y)$.
- Create the kernel matrix $K^{N \times N}$, which has following form

$$\begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{bmatrix}$$

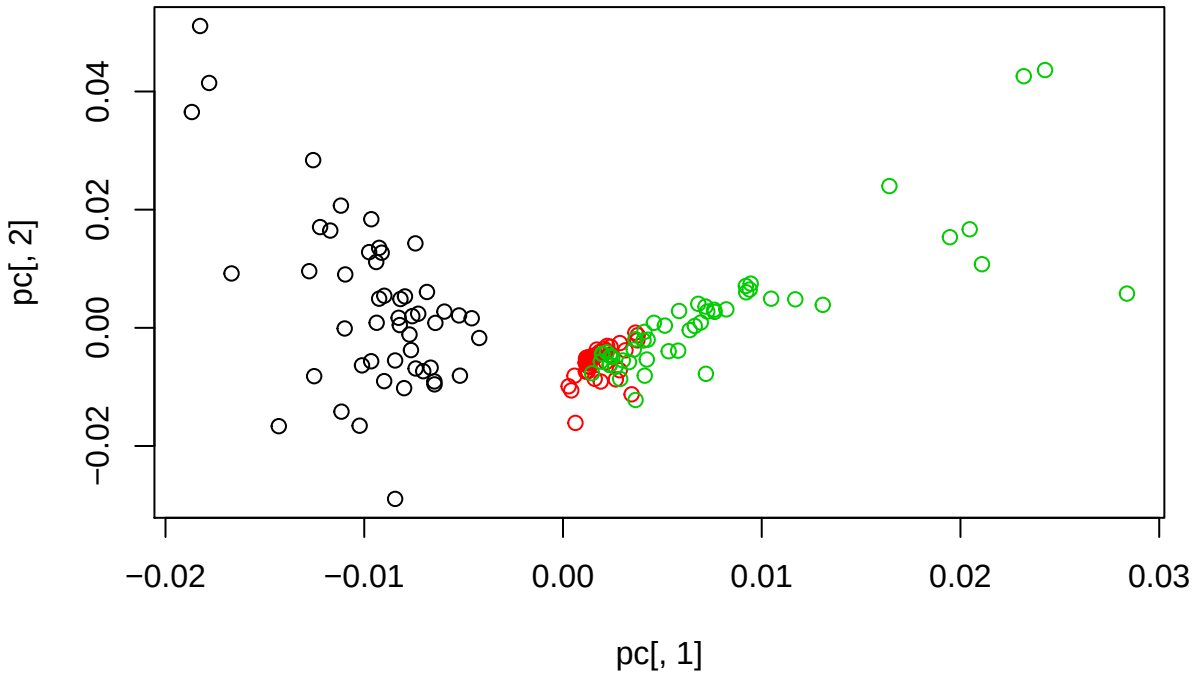
- Center K which results in new matrix $K^\star$.
  - $K^\star = (I_N - M)K(I_N - M)$.
  - $M = 11^T/N$.

- Solve $N\lambda\alpha = K^\star\alpha$
- An equivalent form of the prior eigenvalue problem is $\lambda^\star\alpha = K^\star\alpha$ where $\lambda^\star = N\lambda$.
  - The equivalent form of the eigenvalue problem can be solved using standard programs.
- Standardize each eigenvector α_i by dividing of $\sqrt{\lambda_i}$.

Code example of a using a polynomial kernel with degree=3, scale=1 and offset=1.

```
data <- as.data.frame(apply(as.matrix(iris[,-5]), 2, scale))

kernel_matrix <- function(kernel, data){
  num_obs  <- nrow(data)
  data <- as.matrix(data)
  # make the matrix
  K <- matrix(nrow = num_obs, ncol = num_obs)
  # loop is optimized since the matrix is symmetric.
  for(row in 1:num_obs){
    for(col in row:num_obs){
      K[row, col] <- kernel(data[row,], data[col,])
      K[col, row] <- K[row, col]
    }
  }

  return(K)
}

kernel_pca <- function(kernel_matrix, thres = 0.0001){
  N <- nrow(kernel_matrix)
  # Create the mean operator matrix
  I_1_n <- matrix(1, nrow = N, ncol = N) * 1 / N
  # Create a NxN identity matrix
  I <- diag(N)
  # Center kernel matrix K
  K_star <- (I - I_1_n) %*% K %*% (I - I_1_n)

  # Solve the eigenvalue problem
  de = eigen(K_star)
  eigenvalues <- de$values / N

  # Find the eigenvalues whch are greater than the threshold
  index <- eigenvalues > thres
  eigenvalues <- eigenvalues[index]
  N_comp <- length(eigenvalues)

  # Standardized the eigenvectors
  pc <- t(t(de$vectors[, 1:N_comp]) / sqrt(eigenvalues[1:N_comp]))
  # Rotated PCs
  rpc <- K_star %*% pc
  return(list('pc'=pc, 'rpc'=rpc, 'eigen'=eigenvalues))
}

# Create a kernel function
```

```
poly_kernel <- function(x, y, scale=1, degree=3, offset=1){
  return((scale*(t(x) %*% (y))+offset)^degree)
}
# Create a kernel matrix
K <- kernel_matrix(poly_kernel, data)

# Compute a kernel matrix
KPrinComp <- kernel_pca(K, thres=0.0001)
pc <- KPrinComp$pc
# Make a cool plot
plot(pc[,1], pc[,2], col = iris[, 5])
```




```
# Print eigenvalues
KPrinComp$eigen[1:10]
```

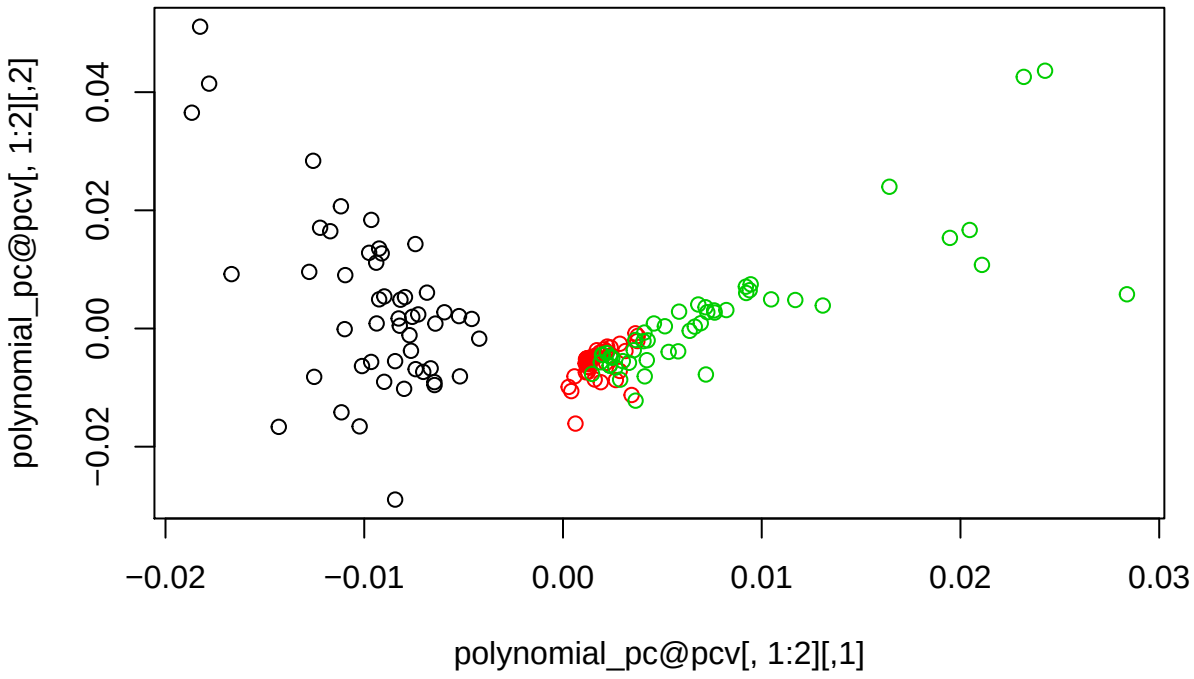
```
##  [1] 97.312668 51.393789 24.161188 15.625179  7.249614  6.833061  4.963342
##  [8]  2.659931  1.733898  1.144715
```

Using the kernlab function

```
library(kernlab)
params <- list(degree = 3, scale = 1, offset = 1)
polynomial_pc <- kpca(~.,data=data, kernel="polydot", kpar = params, th = 0.0001)
polynomial_pc@eig[1:10]
```

```
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7
## 97.312668 51.393789 24.161188 15.625179  7.249614  6.833061  4.963342
##    Comp.8    Comp.9   Comp.10
##  2.659931  1.733898  1.144715
```

```
plot(polynomial_pc@pcv[, 1:2], col = iris[, 5])
```



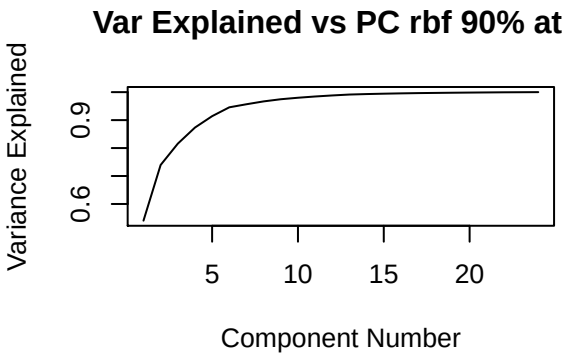
Notes

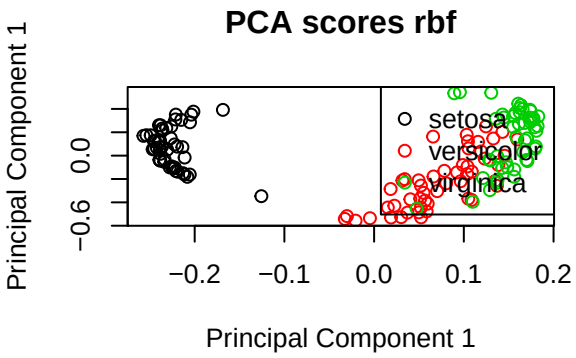
- If we are analyzing a $n \times p$ matrix, linear pca will always yield p principal components.
- With non-linear (kernel pca) we will always produce n eigenvectors and eigenvalues.
  - The number principal components is equall to the number non-zero eigenvalues.
  - In `kernlab`, an extra restiction is set such that all principal components must have eigenvalues with $\lambda > 0.0001$.

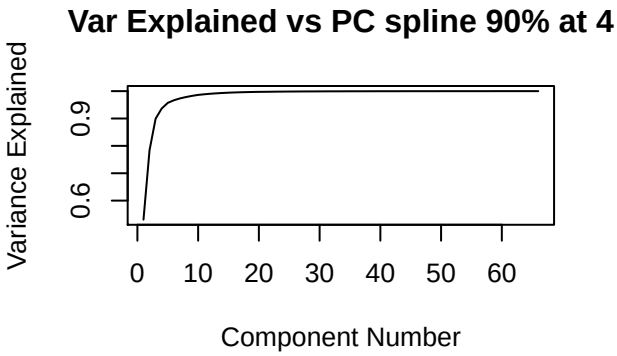
```
library(kernlab)
library(tibble)

graph_legend <- tibble(
  x="topright",
  col= c(1,2,3),
  legend = c("setosa", "versicolor", "virginica"),
  pch = .5
)

pca_plot <- function(kpc, kernel_name){
  var_explained = kpc@eig
  for(i in 2:length(var_explained)){
    var_explained[i] = var_explained[i] + var_explained[i-1]
  }
  var_explained <- var_explained  / var_explained[length(var_explained)]

  pc_number <- min(which(var_explained > .90))

  plot(var_explained,
       type = 'l',
       main = paste('Var Explained vs PC', kernel_name,'90% at',pc_number ),
       xlab = 'Component Number',
```

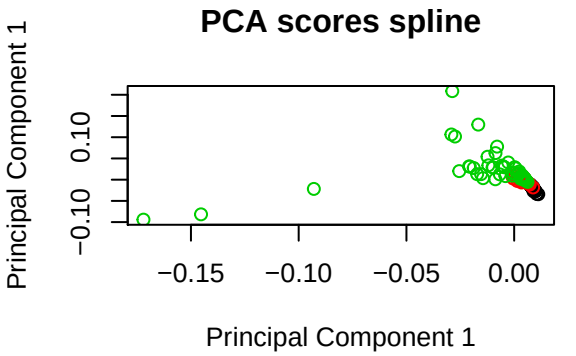
```
       ylab = 'Variance Explained')
}

pca_data_plots <- function(kpc, name, color_vector, graph_legend = NULL){
  #pcv or rotated
  if(length(kpc@eig)>1){
    plot(kpc@pcv[,1:2],
         col = as.numeric(color_vector),
         main = paste('PCA scores', name),
         xlab = 'Principal Component 1',
         ylab = 'Principal Component 1')
  }else{
    plot(kpc@pcv[,1],
         col = as.numeric(color_vector),
         main = paste('PCA scores', name),
         xlab = 'Index',
         ylab = 'Principal Component 1')

  }
  if(!is.null(graph_legend)){

    graphics::legend(graph_legend$x[1],
                     col = graph_legend$col,
                     legend = graph_legend$legend,
                     pch=1)
  }
}
```

```
assign("X", iris[, -5])
X = as.data.frame(apply(X, 2, scale))
gaussian_pc <- kpca(~.,data=X, kernel="rbfdot")
spline_pc <- kpca(~.,data=X, kernel='splinedot', kpar = list())

par(mfrow=c(2,2))
pca_plot(gaussian_pc,'rbf')
pca_data_plots(gaussian_pc, 'rbf',iris[,5], graph_legend)
pca_plot(spline_pc,'spline')
pca_data_plots(spline_pc, 'spline',iris[,5])
```

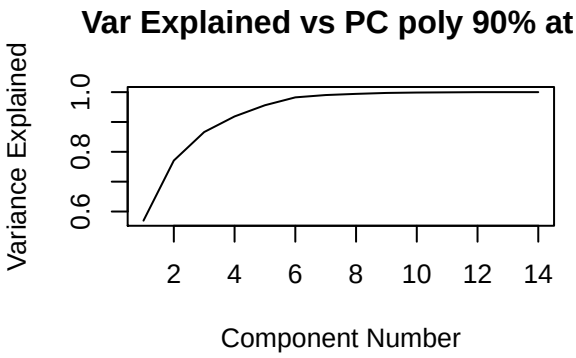


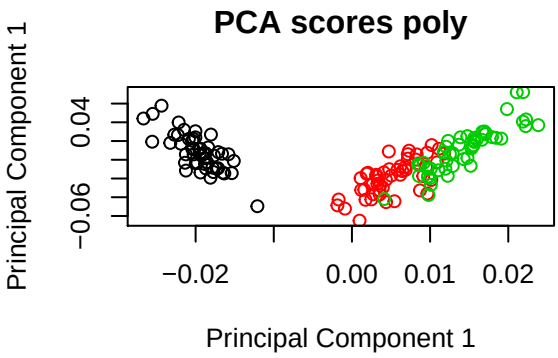


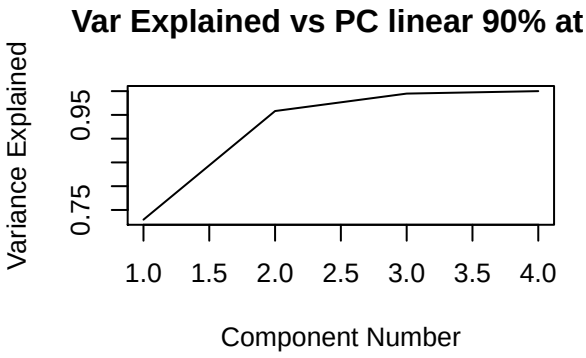


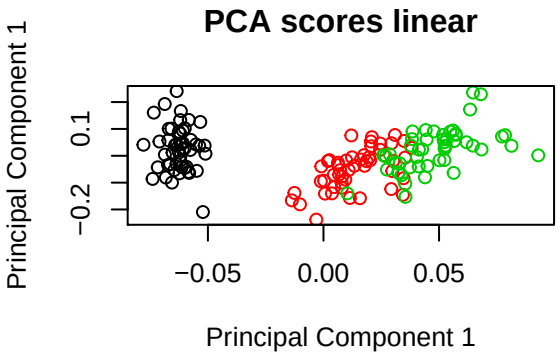



```
linear_pc <- kpca(~.,data=X, kernel="vanilladot", kpar = list())
polynomial_pc <- kpca(~.,data=X, kernel="polydot", kpar = list(degree = 2, scale = 1, offset = 5))

par(mfrow=c(2,2))
pca_plot(polynomial_pc,'poly')
pca_data_plots(polynomial_pc, 'poly', iris[,5])
pca_plot(linear_pc,'linear')
pca_data_plots(linear_pc, 'linear', iris[,5])
```

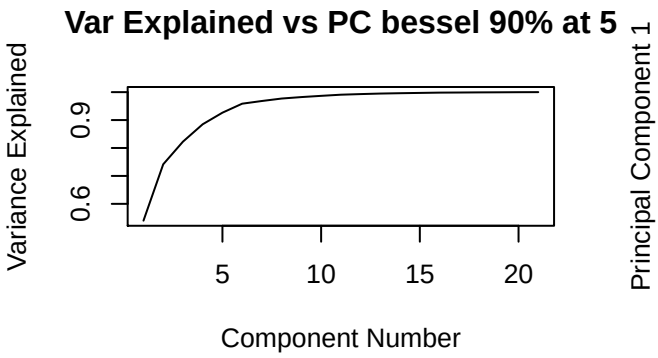


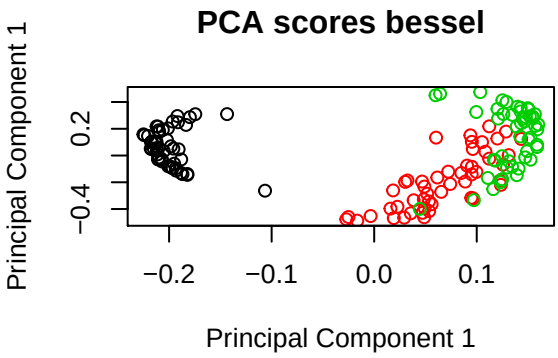


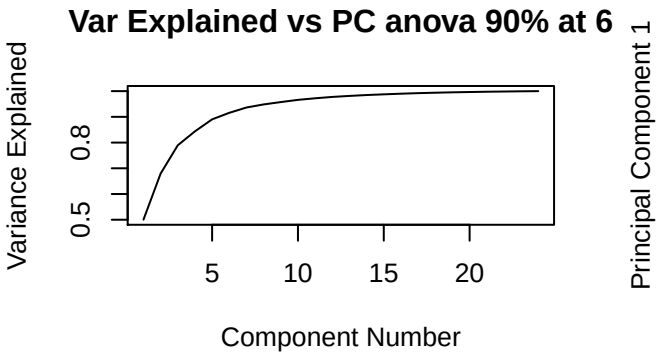


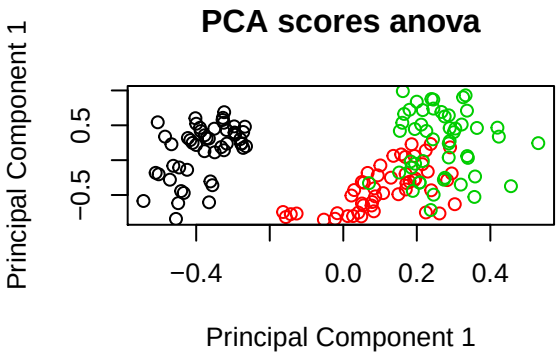



```
bessel_pc <- kpca(~.,data=X, kernel="besseldot", kpar = list(sigma = 1, order = 1, degree = 1))
anova_pc <- kpca(~.,data=X, kernel="anovadot", kpar = list(degree = .1))
par(mfrow=c(2,2))
pca_plot(bessel_pc, 'bessel')
pca_data_plots(bessel_pc, 'bessel', iris[,5])
pca_plot(anova_pc,'anova')
pca_data_plots(anova_pc, 'anova', iris[,5])
```

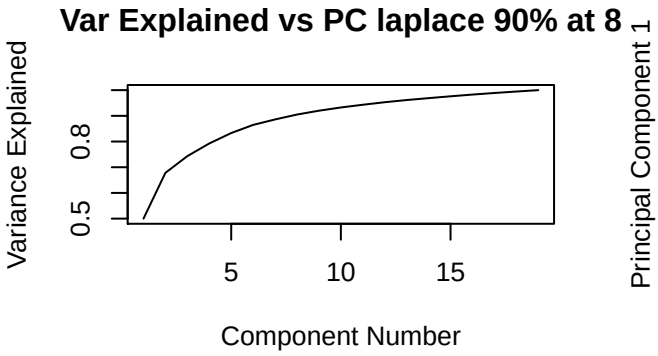


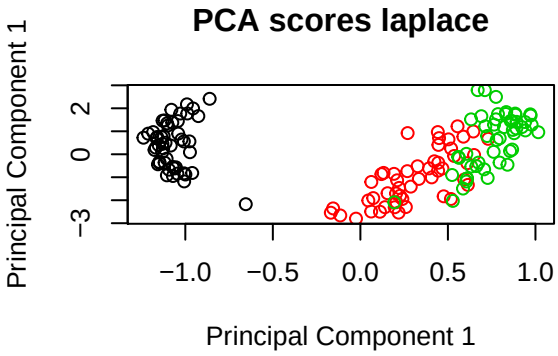


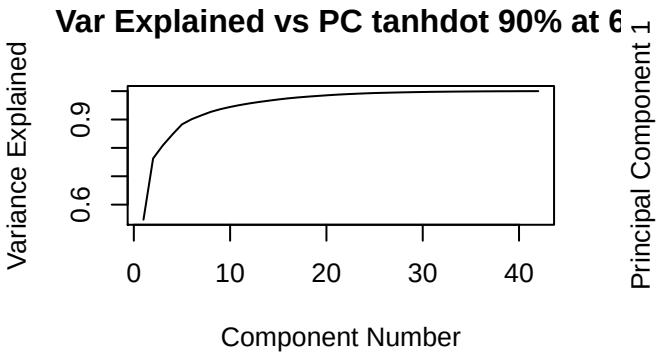


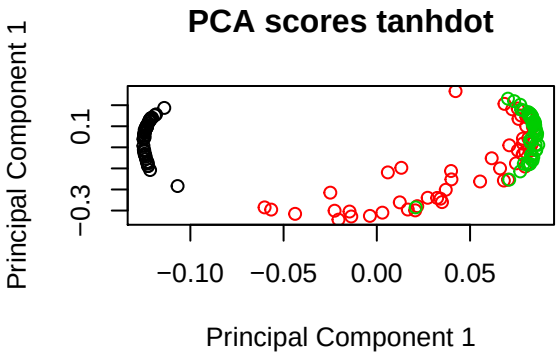



```
laplace_pc <- kpca(~.,data=X, kernel="laplacedot", kpar = list(sigma = 0.01))
tanhdot_pc <- kpca(~.,data=X, kernel="tanhdot", kpar = list(scale = 2, offset = .01))

par(mfrow=c(2,2))
pca_plot(laplace_pc,'laplace')
pca_data_plots(laplace_pc, 'laplace', iris[,5])
pca_plot(tanhdot_pc , 'tanhdot')
pca_data_plots(tanhdot_pc , 'tanhdot', iris[,5])
```



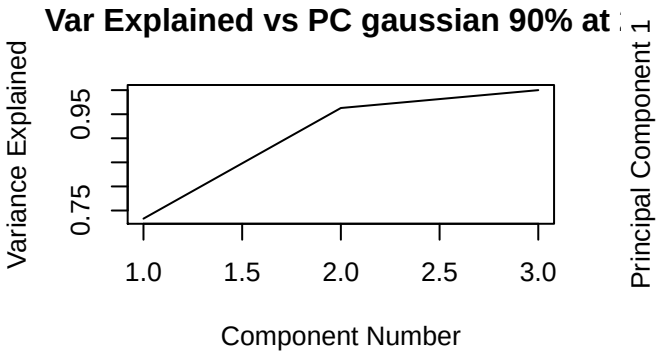


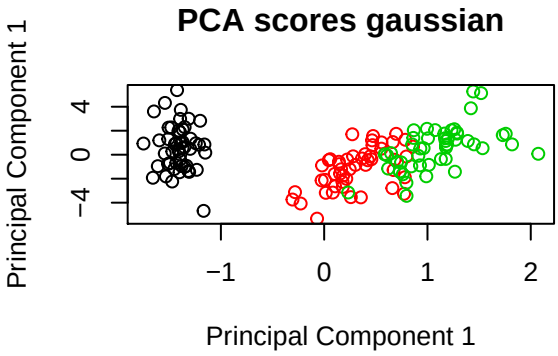


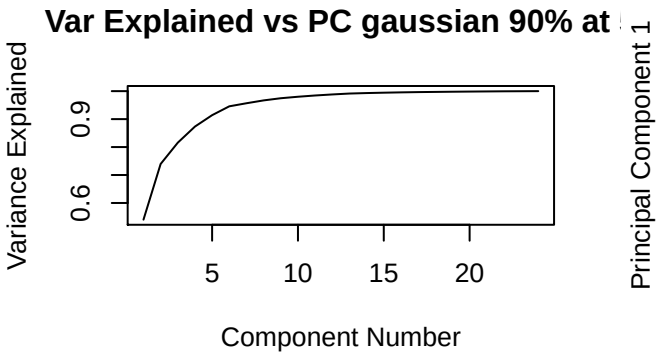


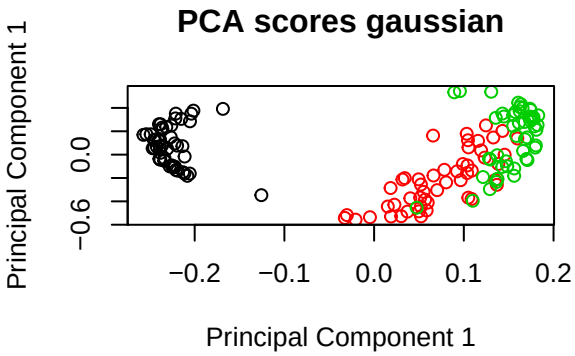
Changing hyperparamters

```
gaussian_pc_modified <- kpca(~.,data=X, kernel="rbfdot", kpar = list(sigma = 0.001))
gaussian_pc_default <- kpca(~.,data=X, kernel="rbfdot", kpar = list(sigma = 0.1))
par(mfrow=c(2,2))
pca_plot(gaussian_pc_modified, 'gaussian')
pca_data_plots(gaussian_pc_modified, 'gaussian', iris[,5])
pca_plot(gaussian_pc_default,'gaussian')
pca_data_plots(gaussian_pc_default, 'gaussian', iris[,5])
```

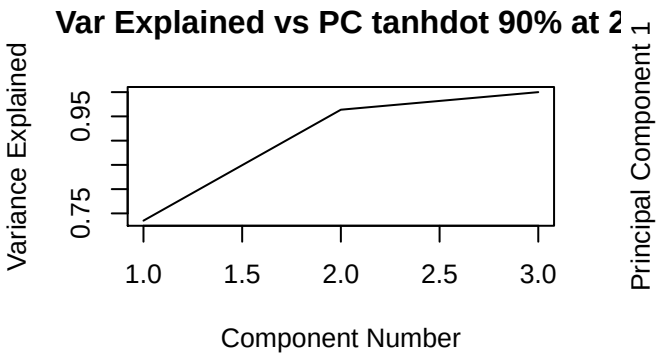


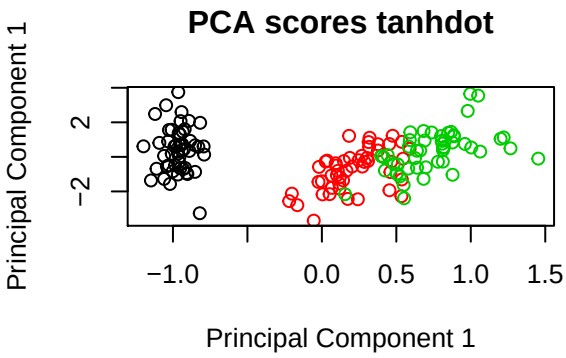


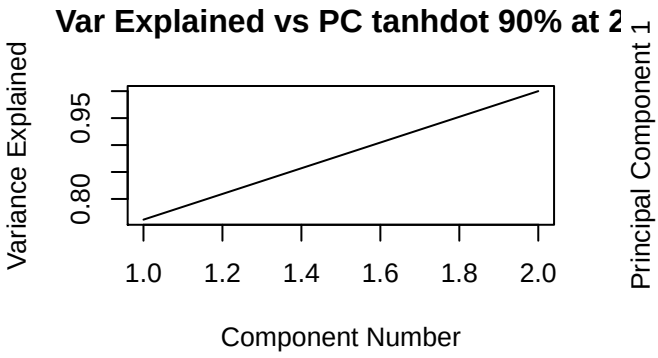


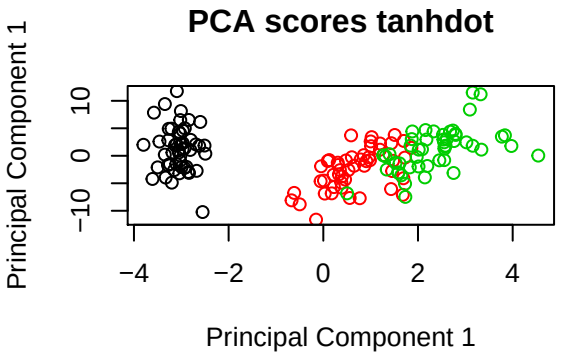



```
tanhdot_pc_1 <- kpca(~.,data=X, kernel="tanhdot", kpar = list(scale = .01, offset = 1))
tanhdot_pc_2 <- kpca(~.,data=X, kernel="tanhdot", kpar = list(scale = .001, offset = 1))

par(mfrow=c(2,2))
pca_plot(tanhdot_pc_1 , 'tanhdot')
pca_data_plots(tanhdot_pc_1 , 'tanhdot', iris[,5])
pca_plot(tanhdot_pc_2 , 'tanhdot')
pca_data_plots(tanhdot_pc_2 , 'tanhdot', iris[,5])
```







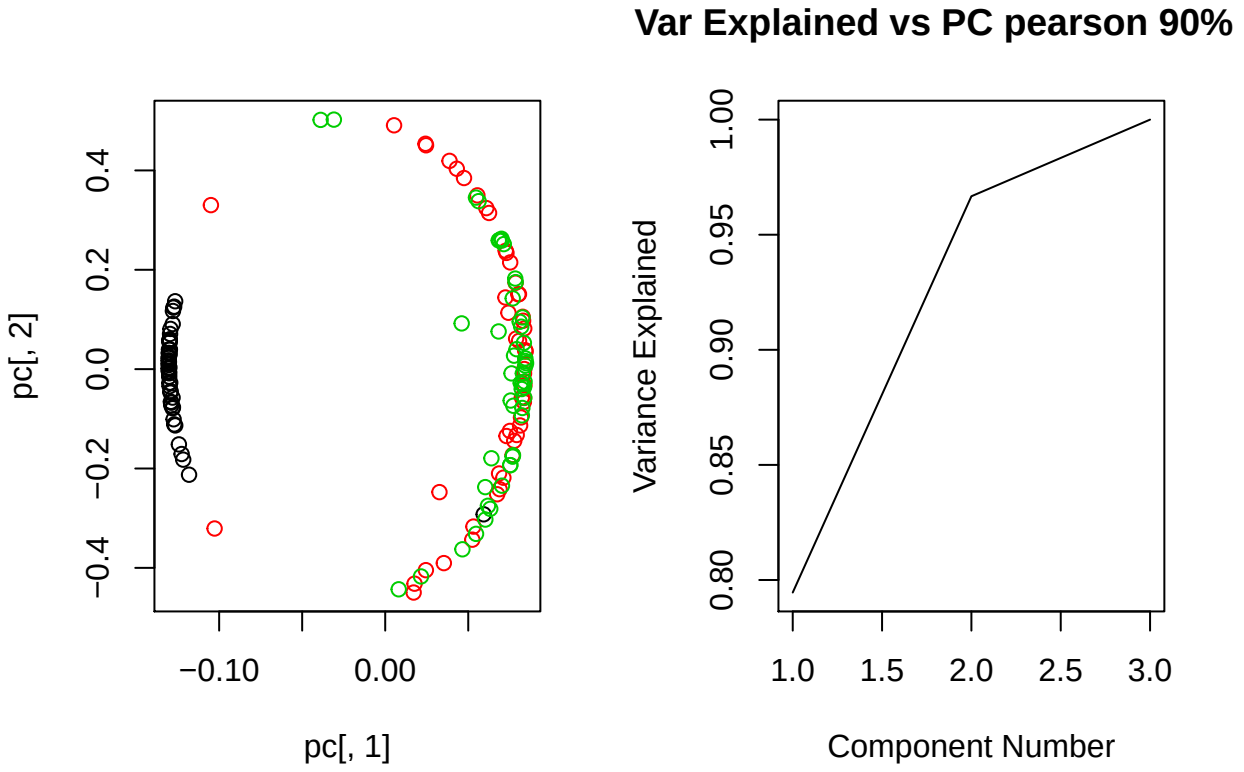


Examples using custom kernels

pearson correlation

```
# Create a kernel function
cor_kernel <- function(x, y){
  return(cor(x,y))
}
# Create a kernel matrix
K <- kernel_matrix(cor_kernel, data)

# Compute a kernel matrix
KPrinComp <- kernel_pca(K, thres=0.0001)
pc <- KPrinComp$pc
# Make a cool plot
par(mfrow=c(1,2))
plot(pc[,1], pc[,2], col = iris[, 5])

var_explained <- KPrinComp$eigen
for(i in 2:length(var_explained)){
  var_explained[i] = var_explained[i] + var_explained[i-1]
}
var_explained <- var_explained  / var_explained[length(var_explained)]

pc_number <- min(which(var_explained > .90))

plot(var_explained,
     type = 'l',
```

```
     main = paste('Var Explained vs PC', 'pearson','90% at',pc_number ),
     xlab = 'Component Number',
     ylab = 'Variance Explained')
```



QWK kernel

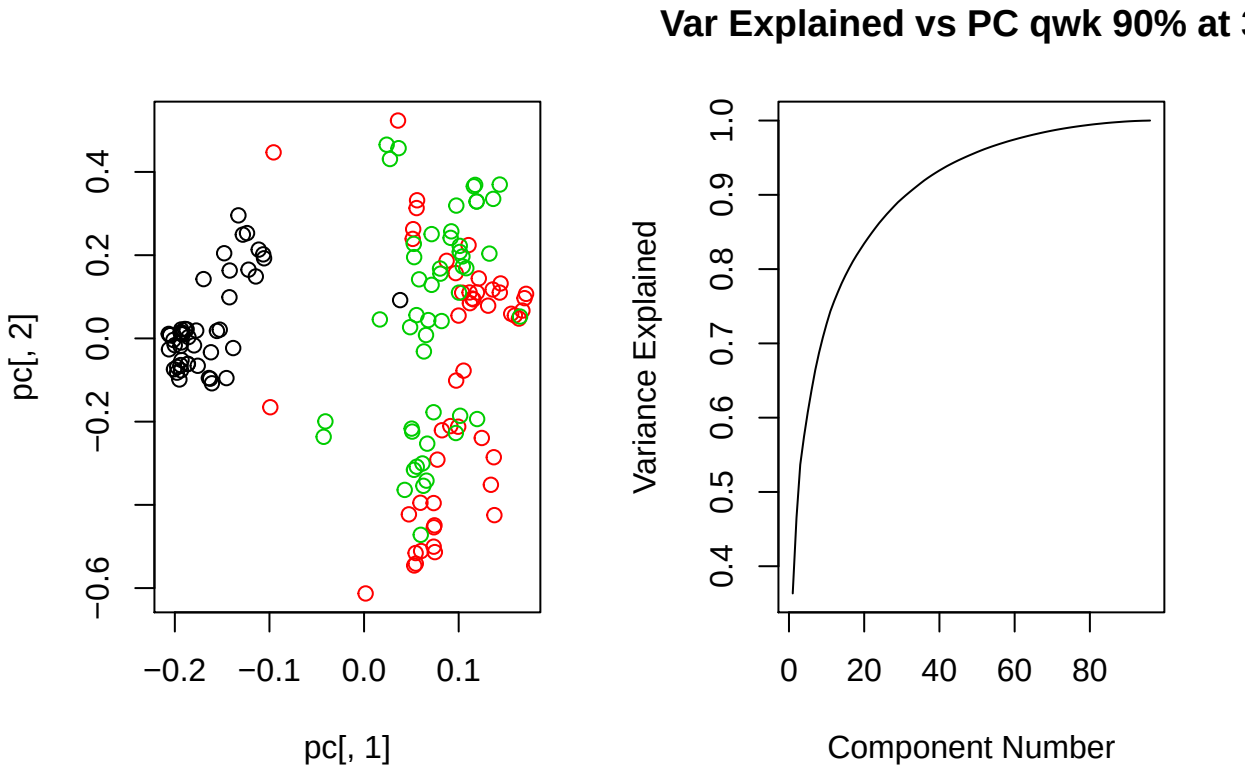
```
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
# Create a kernel function
qwk_kernel <- function(x, y){
  return(cohen.kappa(cbind(x,y))$weighted)
}
# Create a kernel matrix
K <- kernel_matrix(qwk_kernel, data)

# Compute a kernel matrix
KPrinComp <- kernel_pca(K, thres=0.0001)
pc <- KPrinComp$pc
# Make a cool plot
par(mfrow=c(1,2))
plot(pc[,1], pc[,2], col = iris[, 5])

var_explained <- KPrinComp$eigen
```

```r
for(i in 2:length(var_explained)){
  var_explained[i] = var_explained[i] + var_explained[i-1]
}
var_explained <- var_explained  / var_explained[length(var_explained)]

pc_number <- min(which(var_explained > .90))

plot(var_explained,
     type = 'l',
     main = paste('Var Explained vs PC', 'qwk','90% at',pc_number ),
     xlab = 'Component Number',
     ylab = 'Variance Explained')
```



gamma kernel

```r
library(psych)
# Create a kernel function
gamma_kernel <- function(x, y){
  return(sum(dgamma(x-y, 1, .20)))
}
# Create a kernel matrix
K <- kernel_matrix(gamma_kernel, data)

# Compute a kernel matrix
KPrinComp <- kernel_pca(K, thres=0.0001)
pc <- KPrinComp$rpc
# Make a cool plot
par(mfrow=c(1,2))
```

```r
plot(pc[,1], pc[,2], col = iris[, 5])

var_explained <- KPrinComp$eigen
for(i in 2:length(var_explained)){
  var_explained[i] = var_explained[i] + var_explained[i-1]
}
var_explained <- var_explained  / var_explained[length(var_explained)]

pc_number <- min(which(var_explained > .90))

plot(var_explained,
     type = 'l',
     main = paste('Var Explained vs PC', 'gamma','90% at',pc_number ),
     xlab = 'Component Number',
     ylab = 'Variance Explained')
```