Kernel PCA Portfolio

Rachel Wood

2023-02-10

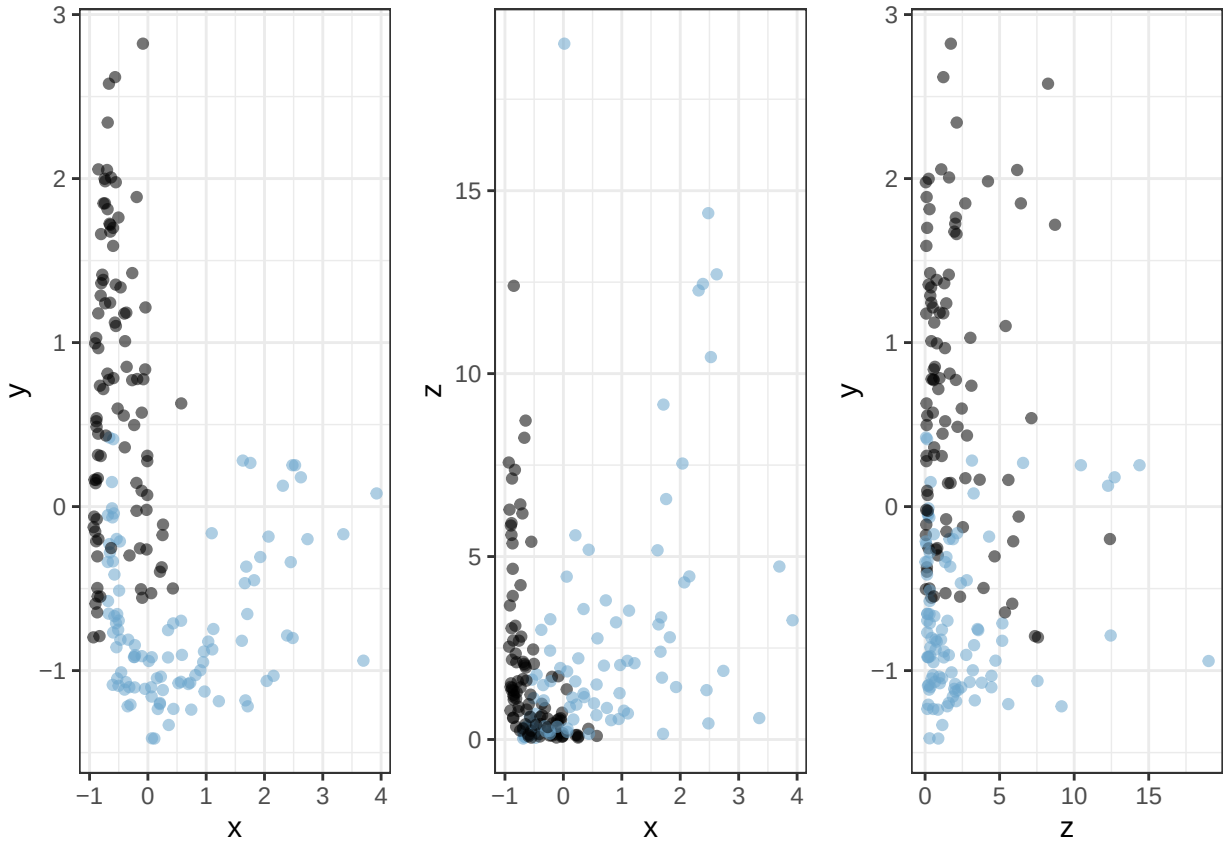
For this portfolio, I will make use of the `clusteringdatasets` package which contains a `make_moons()` function. We use this to create moon shapes, adding a third dimension of the row number. We also add noise and take the exponential of the values to add more non-linearity.

We load, scale and plot the data below:

```
library(clusteringdatasets)
library(gridExtra)
moons <- make_moons(n_samples = 200, shuffle = FALSE, noise = 0.2)
z <- as.vector(exp(scale(1:200) + rnorm(200)))

df <- scale(exp(moons$samples))
df <- cbind(df, moons$labels, order = 1:100)

df <- df %>%
  as_tibble() %>%
  dplyr::rename("x" = "V1",
                "y" = "V2",
                "label" = "V3") %>%
  arrange(order) %>%
  mutate(z = z,
         label = as.factor(label)) %>%
  select(- order)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
## `.name_repair` is omitted as of tibble 2.0.0.
## i Using compatibility `.name_repair`.
```

```
p1 <- ggplot(data = df, aes(x =x, y=y, color = label, alpha = 0.7)) +
  geom_point() +
  theme(legend.position = "none", )

p2 <- ggplot(data = df, aes(x = x, y=z, color = label, alpha = 0.7))+
  geom_point() +
  theme(legend.position = "none")

p3 <- ggplot(data = df, aes(x = z, y = y, color = label, alpha = 0.7)) +
  geom_point() +
  theme(legend.position = "none")

grid.arrange(p1, p2, p3 , ncol = 3)
```



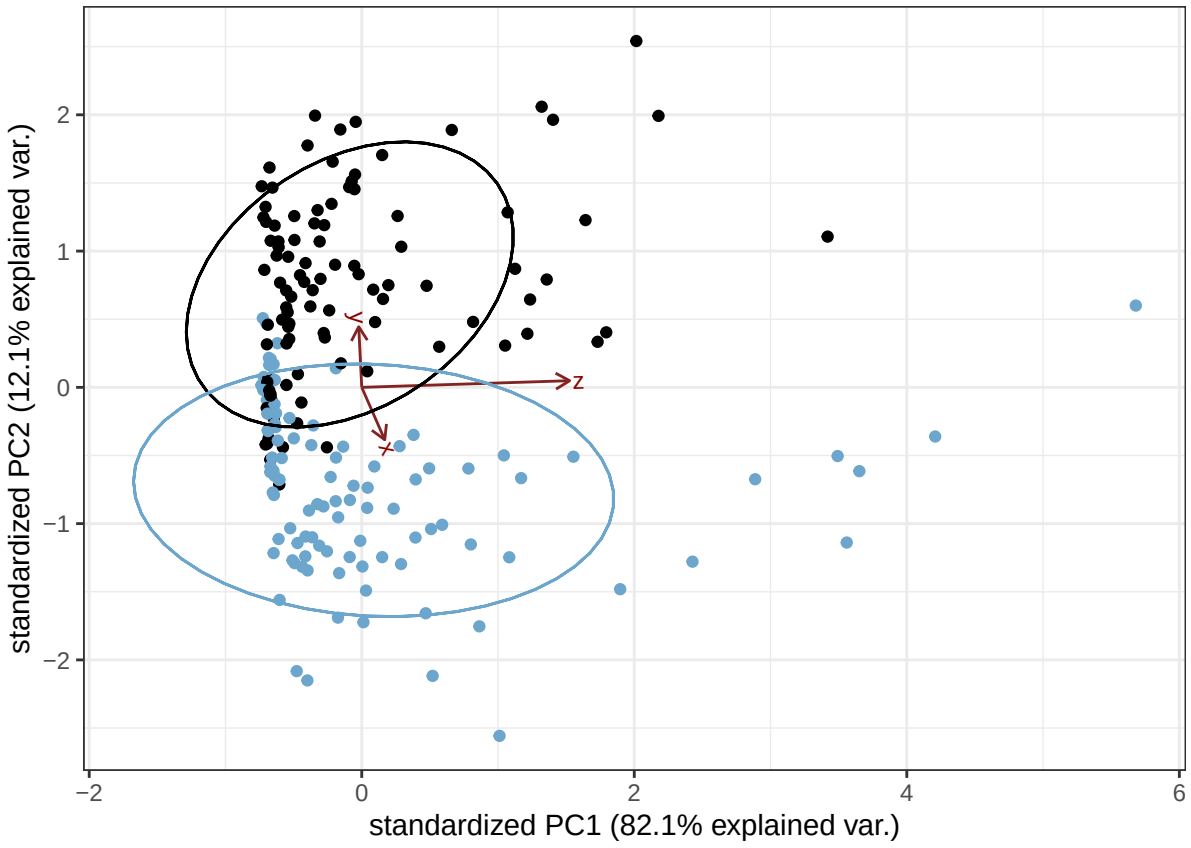
We can see these are not linearly separable in any plane, and thus we expect fitting a logistic regression model on the data would produce poor results.

```
train <- sample(1:200, 150)
df_train <- df[train,]
df_test <- df[-train,]

original_fit <- glm(label ~ x + y + z, data = df_train, family = binomial(link = "logit"))

prediction <- round(predict.glm(original_fit, df_test, type = "response")) +1

sum(df_test$label == prediction )/50
```

```
## [1] 0.72
```

Indeed we can see this only predicts the correct value 72% of the time, which is not terrible, but we see if we can use PCA to obtain a better result. # Standard PCA We can see that this data forms 3 distinct groups, but are not linearly separable. For this reason, we do not expect standard PCA to work well for this data. The following analysis shows this intuition is correct:

```
library(ggbiplot)
pca <- prcomp(~x + y +z, data = df)

ggbiplot(pca, groups = df$label, ellipse = TRUE) +
  theme(legend.position = "none")
```



As we can see, our principal components have transformed the data, but the two groups are still not linearly separable, hence we use kernel PCA.

Kernel PCA

We now perform kernel PCA, with a variety of kernels to determine the best choice. We will consider:

- A polynomial kernel
- A Gaussian kernel

Polynomial Kernel

The polynomial kernel for a degree d is defined by

$$k_d(x, x') = \Phi_d(x)^T \Phi_d(x'), \qquad (x, x') \in \mathbb{R}^p \times \mathbb{R}^p$$

with $\Phi_d(x) = \{1, x, x^2, \ldots, x^d\}$.

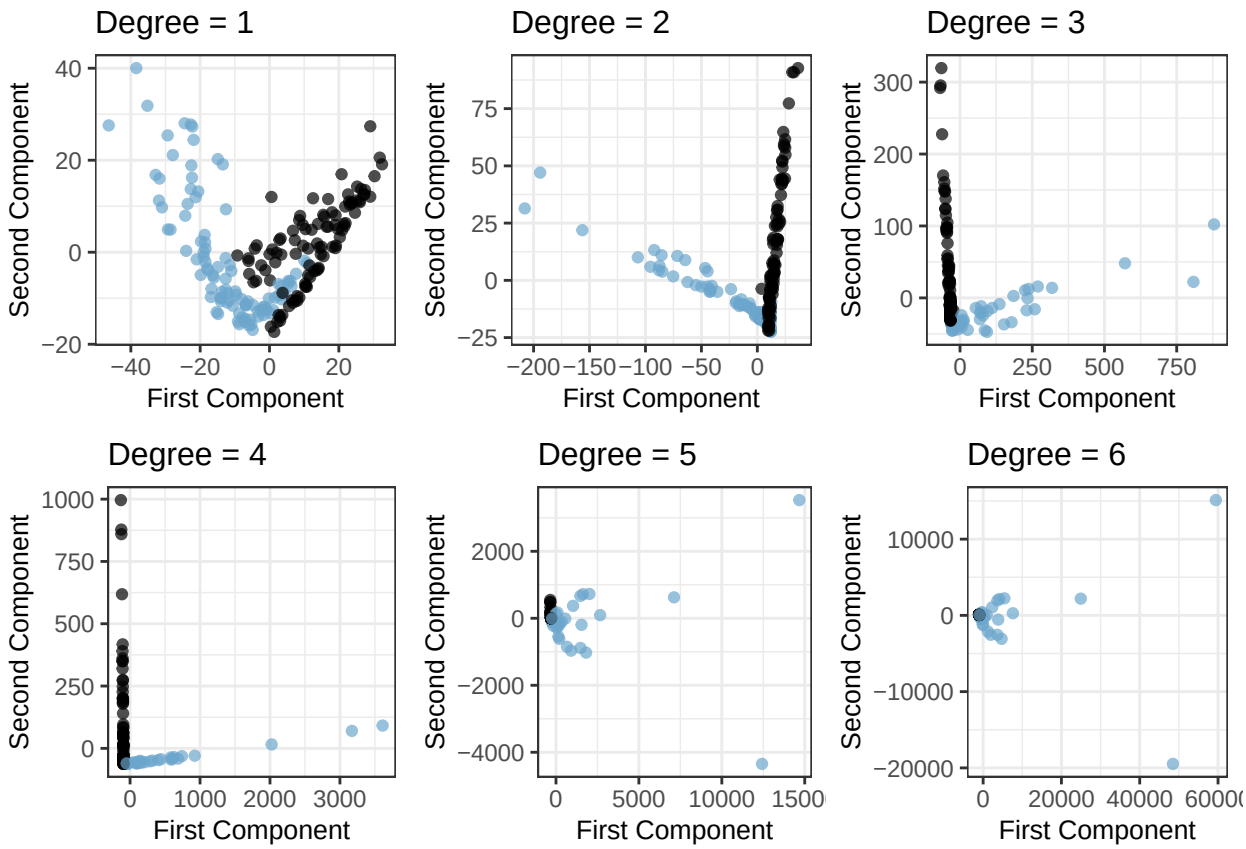
The following code fits obtains the first two principal components obtained by performing kernel PCA and plots the results of using $d = 1, \ldots, 6$:

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

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

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
degree <- 1:6


plot_list <- vector(mode = "list", length = length(degree))

for (i in 1:length(degree)){
  kpca <- kpca(~x + y, data = df, kernel = "polydot", kpar = list(degree = degree[i]))
  result <- predict(kpca, df)

  kpca_points <- result[,1:2] %>%
    as_tibble() %>%
    mutate(label =df$label)
  colnames(kpca_points) <- c("x", "y", "label")

  p <- ggplot(data = kpca_points, aes(x,y, color = label) ) +
    geom_point(position = "jitter", alpha = 0.7) +
    theme(legend.position = "none", title = element_text(size = 10)) +
    labs(x = "First Component", y = "Second Component", title = paste("Degree =", degree[i]))
  plot_list[[i]] <- p
}

do.call("grid.arrange", c(plot_list, ncol = 3))
```



We can see the polynomial kernel seperates the data well enough, but it is limited to a finite dimensional feature space - which we have to specify.

Gaussian Kernel

We thus try a Gaussian (or an rbf) kernel which is defined by:

$$k_\sigma(x, x') = \exp\left(-\frac{||x - x'||^2}{\sigma^2}\right)$$

A common practice is setting $\sigma = \text{median}\{||x_i - x_j||^2\}$, which in our case is:

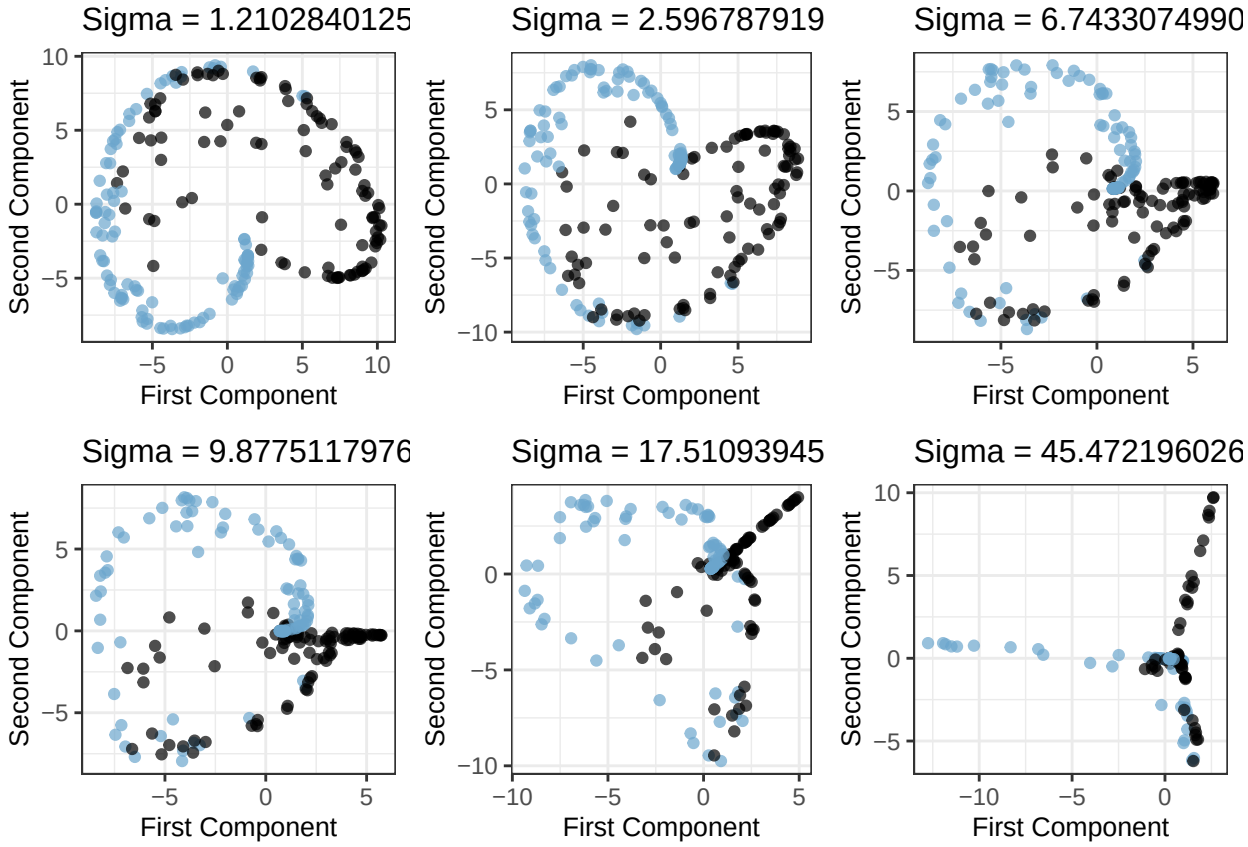
```
library(rdist)

sigma <- median(as.vector(pdist(df[c(1:2,4)]))^2)
sigma
```

```
## [1] 6.743307
```

The following code fits a

```
sigma_vec <- sigma ^ c(0.1,0.5, 1,1.2,1.5,2)

model_list <- vector(mode = "list", length = length(sigma_vec))
plot_list <- vector(mode = "list", length = length(sigma_vec))
```

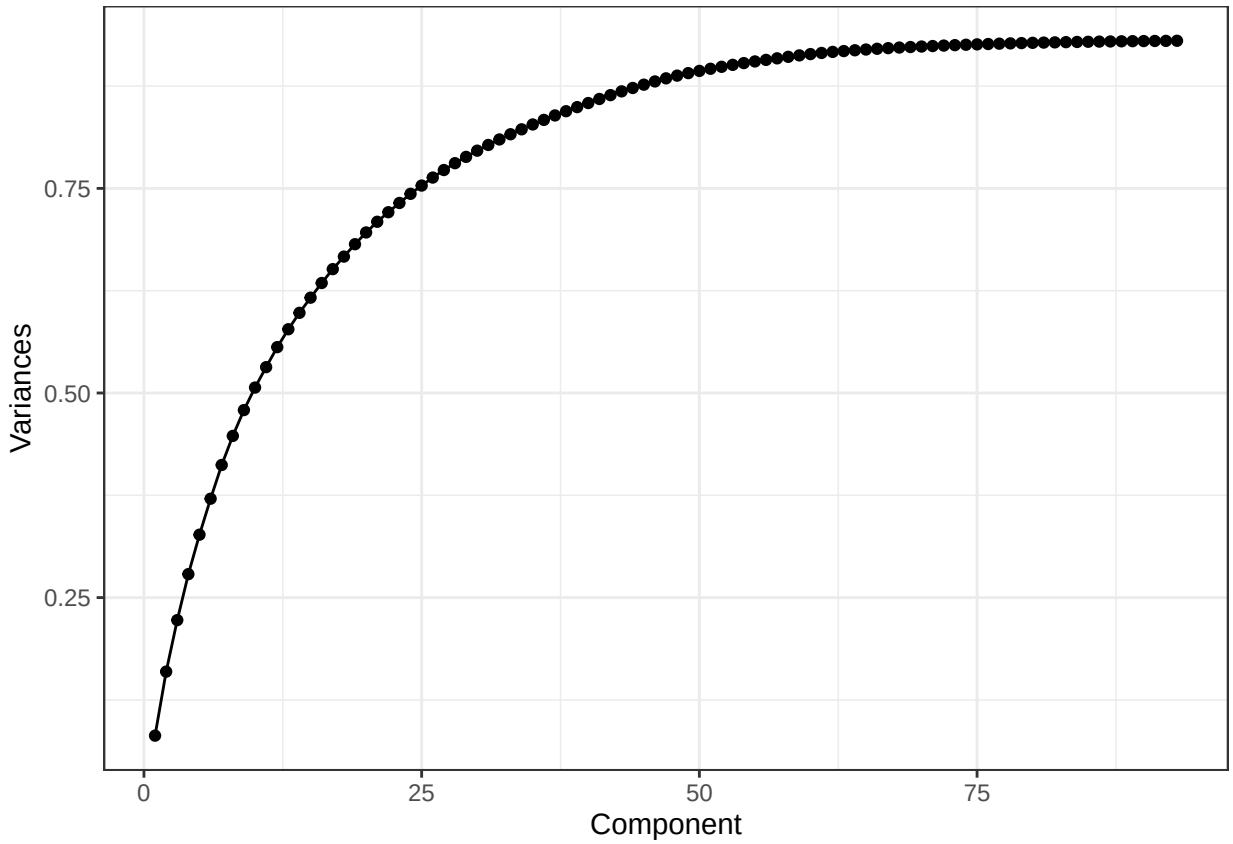
```
for (i in 1:length(sigma_vec)){
  kpca <- kpca(~x + y, data = df, kernel = "rbfdot", kpar = list(sigma = sigma_vec[i]))
  result <- predict(kpca, df)
  model_list[[i]] <- kpca

  kpca_points <- result[,1:2] %>%
    as_tibble() %>%
    mutate(label =df$label)
  colnames(kpca_points) <- c("x", "y", "label")

  p <- ggplot(data = kpca_points, aes(x,y, color = label) ) +
    geom_point(position = "jitter", alpha = 0.7) +
    theme(legend.position = "none", title = element_text(size = 10)) +
    labs(x = "First Component", y = "Second Component", title = paste("Sigma =", sigma_vec[i]))
  plot_list[[i]] <- p
}

do.call("grid.arrange", c(plot_list, ncol = 3))
```



We can see the embedding with the value of `sigma` obtained using the 'median' trick provides a reasonable embedding and thus we continue with this value. # Logistic Regression with KPCA

We now fit a logistic regression using the components given by the Gaussian kernel. First we do a scree plot to determine an appropriate number of components

```
accuracy <- vector(length = length(sigma_vec))
model <- model_list[[3]]
variances <- cumsum(eig(model))
screedat <- tibble("Component" = 1:length(variances), "Variances" = variances)

ggplot(screedat, aes(x = Component, y = Variances)) +
  geom_point() +
  geom_line()
```



We can see that after 60 components, there is not much improvement in the proportion of variance explained and so we keep the first 60 components:

```
model <- model_list[[3]]
features <- rotated(model)
features <-cbind("label" = df$label, features[,1:60]) %>%
  as_tibble() %>%
  mutate(label = as.factor(label))

features_train <- features[train,]
features_test <- features[-train,]

logistic_model <- glm(formula = label ~. , data = features_train, family = binomial)

predicted <- round(predict.glm(logistic_model, features_test, type = "response")) +1
sum(features_test$label == predicted )/50
```

```
## [1] 0.94
```

We get a 94% success rate, a significant improvement on the classification performed on the original data.