Statistical Methods 2: Portfolio 3

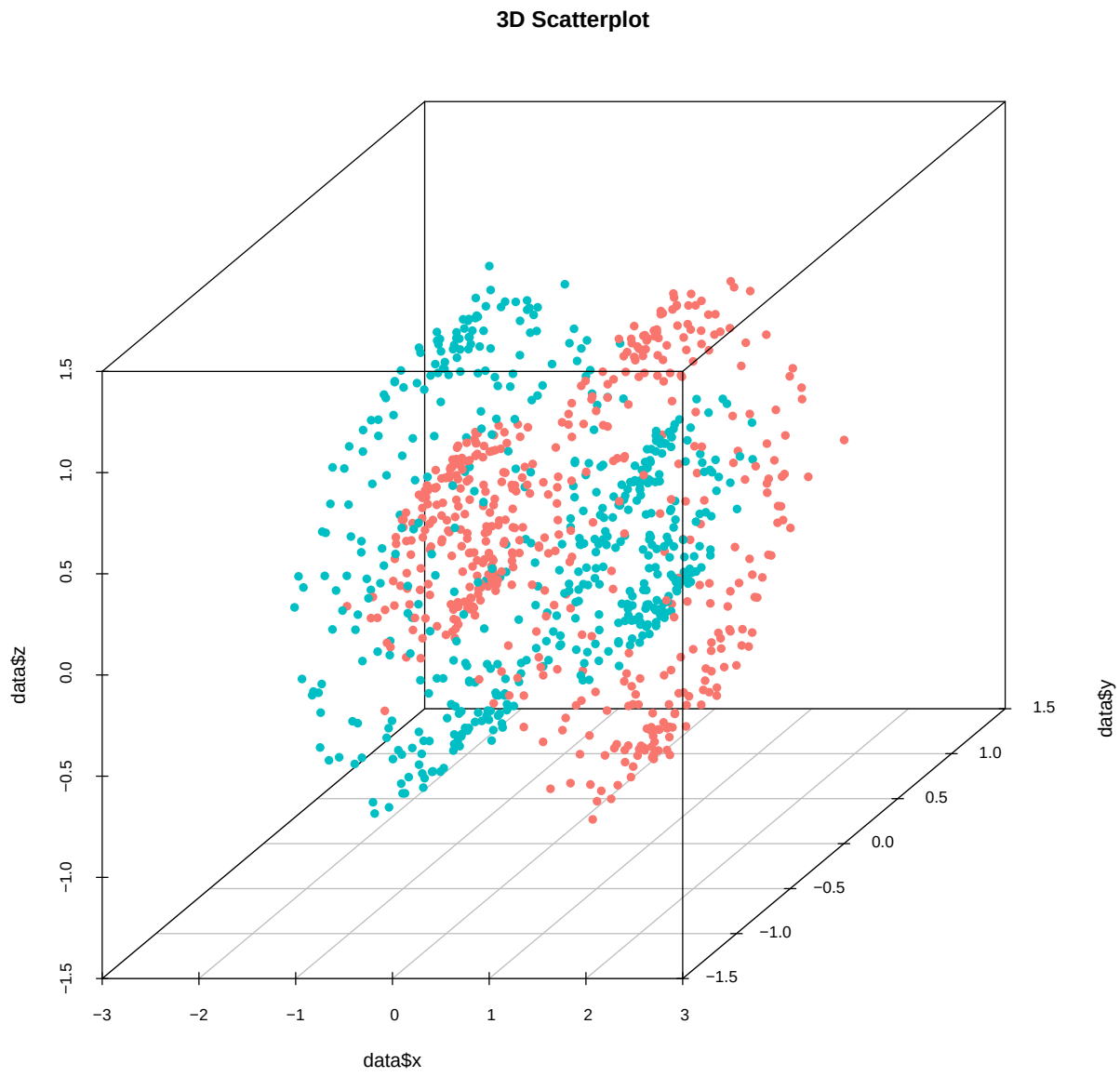
Kieran Morris

The Data

Inspired by the dataset from the lecture notes, we add an inverted friend to the dataset, although the friend has its cluster types inverted. So our task it to classify the shell of one sphere and the core of another. Kind of like a bad 3D 'Ying-Yang' symbol. Below we generate and plot the pure 3D data, it's a bit hard to make out as we don't have interactivity.

```
library(scatterplot3d)
library(ggplot2)
library(gridExtra)
library(ggpubr)
library(ggbiplot)

# Define a function to generate points on a sphere
sphereFun <- function(center = c(0,0,0), diameter = 1, npoints = 200){
  tt <- runif(npoints, 0, 2*pi)
  phi <- runif(npoints, 0, pi)
  xx <- center[1] + diameter * sin(phi) * cos(tt)
  yy <- center[2] + diameter * sin(phi) * sin(tt)
  zz <- center[3] + diameter * cos(phi)
  return(data.frame(x = xx, y = yy, z = zz))
}

# Generate points for four spheres with different centers and radii
sphere1 <- sphereFun(center = c(-1,0,0), diameter = 0.4, npoints = 200)
sphere2 <- sphereFun(center = c(-1,0,0), diameter = 1, npoints = 300)
sphere3 <- sphereFun(center = c(1,0,0), diameter = 0.4, npoints = 200)
sphere4 <- sphereFun(center = c(1,0,0), diameter = 1, npoints = 300)
# Add random noise to the points
noise <- 0.1
sphere1 <- sphere1 + rnorm(nrow(sphere1), sd = noise)
sphere2 <- sphere2 + rnorm(nrow(sphere2), sd = noise)
sphere3 <- sphere3 + rnorm(nrow(sphere3), sd = noise)
sphere4 <- sphere4 + rnorm(nrow(sphere4), sd = noise)

# Combine the datasets into one and add a cluster column
data1 <- data.frame(sphere1, cluster = '0')
data2 <- data.frame(sphere2, cluster = '1')
data3 <- data.frame(sphere3, cluster = '1')
data4 <- data.frame(sphere4, cluster = '0')

data <- rbind(data1, data2, data3, data4)
# Create a color vector based on the cluster column for plotting
```

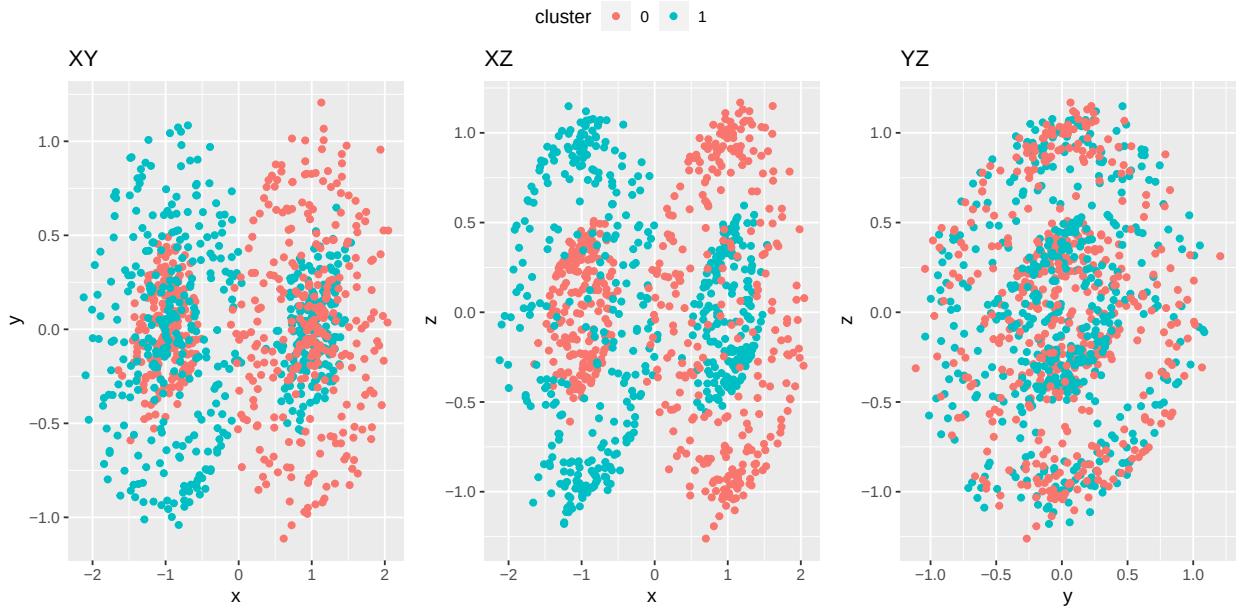
```r
color_vector <- ifelse(data$cluster == '0', '#F8766D', '#00BFC4')

# Plot the 3D data with the color vector
scatterplot3d(data$x, data$y, data$z, color = color_vector, pch = 16, main = '3D Scatterplot')
```



Below we project the data into 2D, this is not with PCA, simply by taking pairs of dimensions and projecting.

```r
# Plot xy projection
PLt_xy <- ggplot(data, aes(x = x, y = y, color = cluster)) +
  geom_point() +
  scale_color_manual(values = c('0' = '#F8766D', '1' = '#00BFC4')) +
  ggtitle('XY')
# Plot xz projection
Plt_xz <- ggplot(data, aes(x = x, y = z, color = cluster)) +
```

```
  geom_point() +
  scale_color_manual(values = c('0' = '#F8766D', '1' = '#00BFC4')) +
  ggtitle('XZ')
# Plot yz projection
Plt_yz <- ggplot(data, aes(x = y, y = z, color = cluster)) +
  geom_point() +
  scale_color_manual(values = c('0' = '#F8766D', '1' = '#00BFC4')) +
  ggtitle('YZ')
# Plot the 2D projections in a grid
ggarrange(Plt_xy, Plt_xz, Plt_yz, ncol = 3,common.legend = TRUE)
```



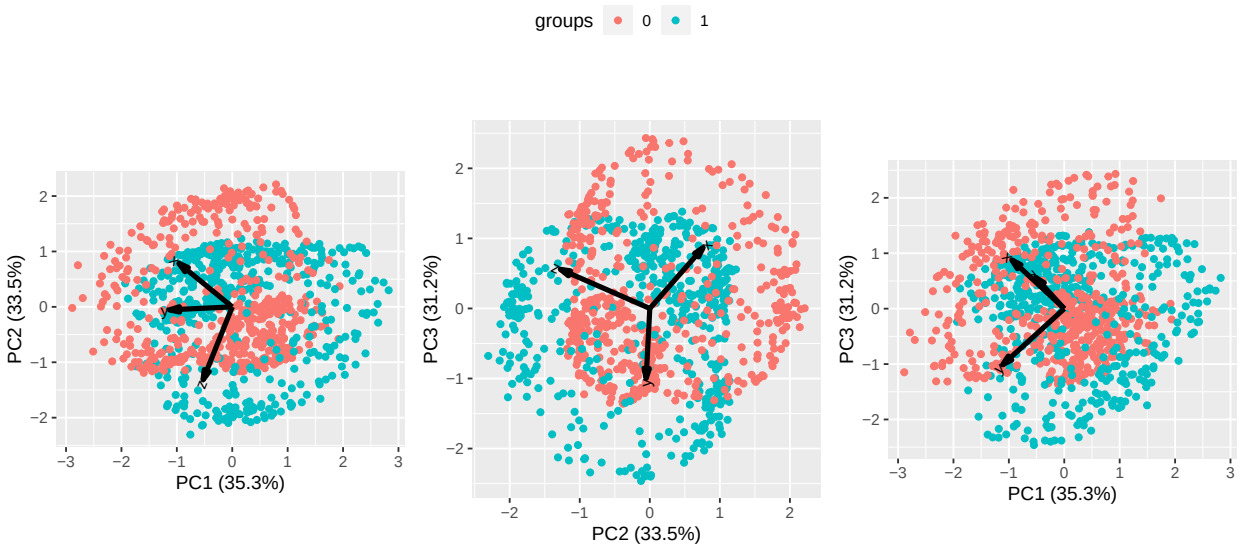
These are much neater looking! Although notice that the clusters are not linearly seperable at all.

Standard PCA

We suspect that normal PCA will not be sufficient for classification and that all principle components will contribute very similar amounts to variance and be approximately the different axes. For completeness we will perform PCA anyway.

```
# Perform PCA on the data
pca <- prcomp(data[,1:3], scale = TRUE)

# Plot the principal components
Plt_PC12 <- ggbiplot(pca, choices = c(1, 2),
 obs.scale = 1,
 var.scale = 1,
 groups = data$cluster,
 ellipse = FALSE,
 circle = FALSE)
Plt_PC23 <- ggbiplot(pca, choices = c(2, 3),
obs.scale = 1,
```

```
var.scale = 1,
groups = data$cluster,
ellipse = FALSE,
circle = FALSE)
Plt_PC13 <- ggbiplot(pca, choices = c(1, 3),
obs.scale = 1,
var.scale = 1,
groups = data$cluster,
ellipse = FALSE,
circle = FALSE)

ggarrange(Plt_PC12, Plt_PC23, Plt_PC13, ncol = 3, common.legend = TRUE)
```



As expected standard PCA gave us nothing, and the results are remarkably similar to the 2D projections. As we said this is expected since variance is fairly equally distributed for each class.

Kernel PCA

Polynomial PCA

Now that we've binned off standard PCA let's try a kernel. Since our data is a cousin of the concentric balls data from the notes, we could apply a similar transform - note that the original data could be seperated with the following feature transform:

$$\Phi : (x, y, z) \to (x, y, z, x^2 + y^2 + z^2)$$

as it embeds it on a 4D cone, which is linearly seperable. This can also be viewed as training over a quadratic kernel. Unfortunately considering the multinomial we would need could get very complex and might cause a very large dimensional feature space, we instead resort to a polynomial kernel , beginning with degree 4. Recall that the polynomial kernel is as follows:

$$K(x, y) = (x^T y + c)^d, \text{ where } c \geq 0 \text{ and } d \in \{4, 5, 6, 7, 8, 9, 10, 11, 13\}.$$

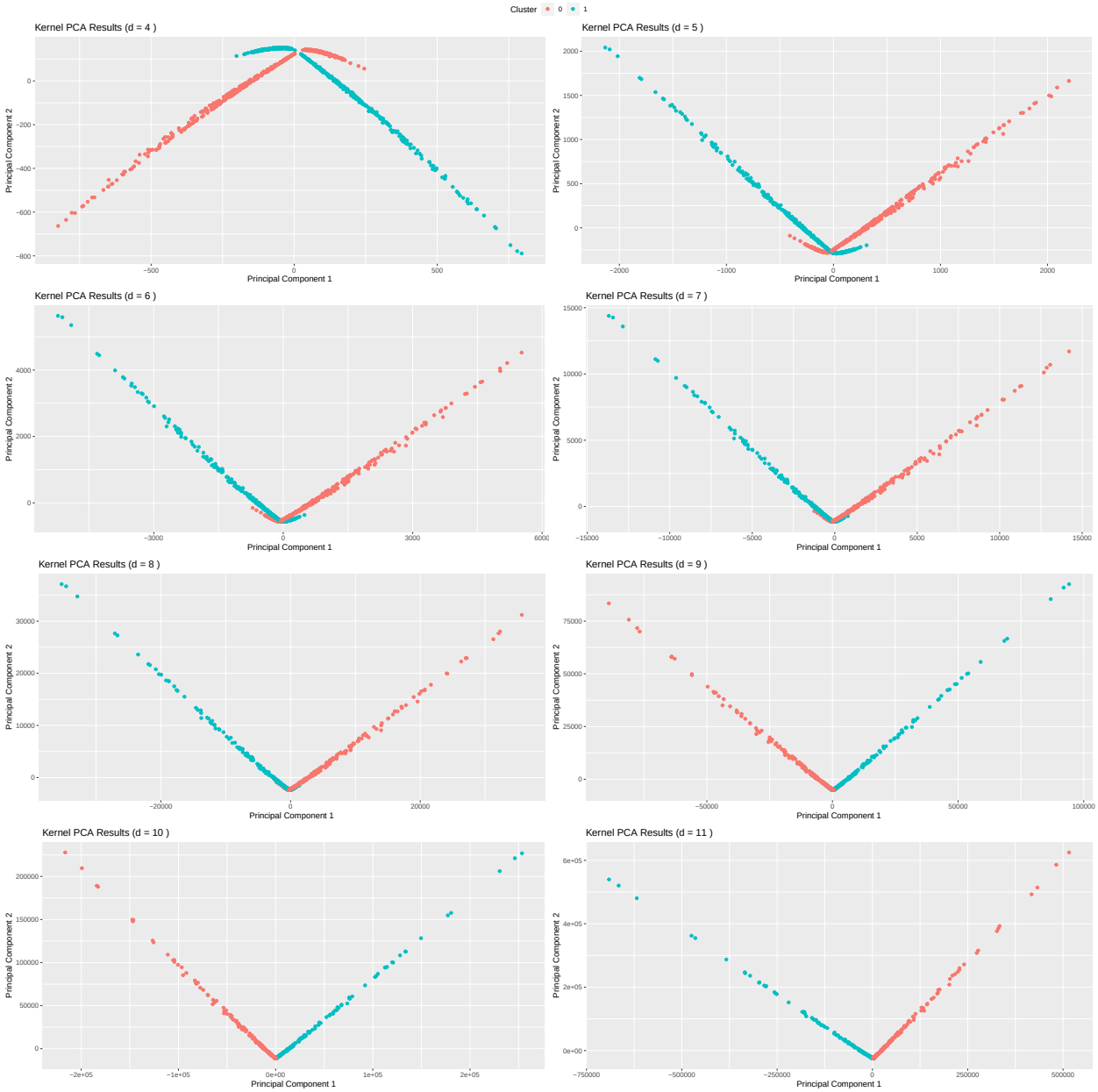
This allows us to have multiple peaks (to seperate the centers) rather than one like in the original example. Let's impliment this in R.

```
library(kernlab)
```

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

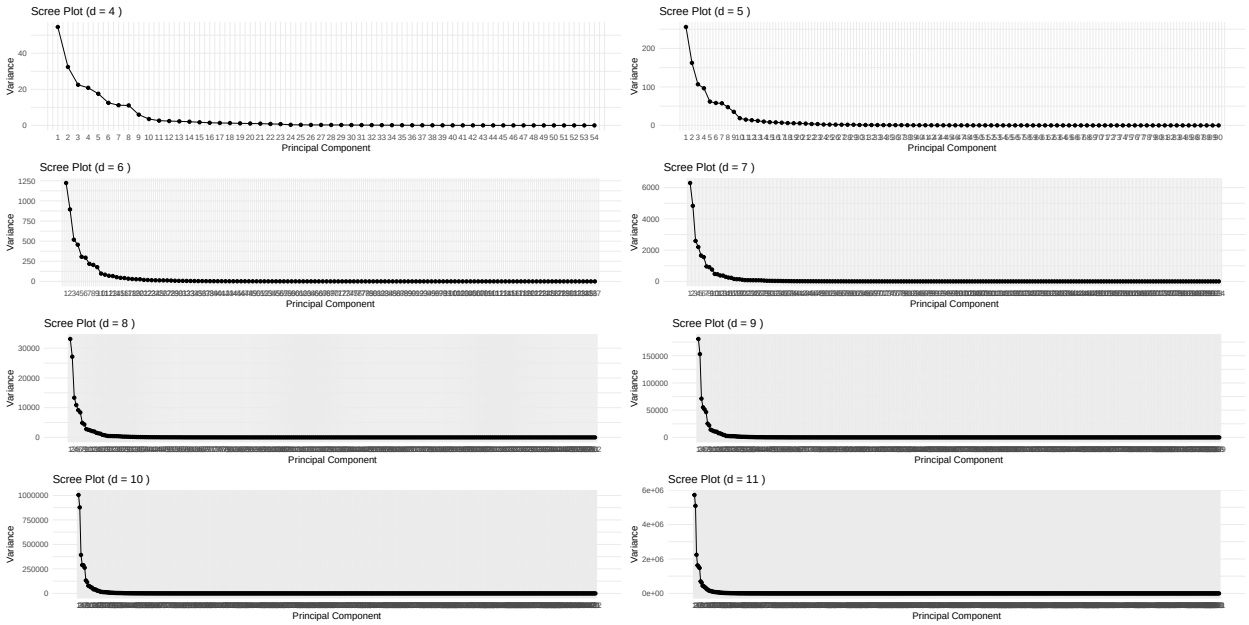
```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
#Create empty lists to store plots and data
plot_list <- list()
kpca_list <- list()
skree_list <- list()
# Loop over the possible d values
for (d in c(4, 5, 6, 7, 8, 9, 10, 11)) {
  # Perform kernel PCA with a polynomial kernel of degree d
  kpca_result <- kpca(~., data = data, kernel = "polydot", kpar = list(degree = d, scale = 1, offset = 

  # Extract the principal components
  pc1 <- kpca_result@rotated[,1]
  pc2 <- kpca_result@rotated[,2]
  pc3 <- kpca_result@rotated[,3]
  pc4 <- kpca_result@rotated[,4]
  #Extract eigenvalues for skree plot
  eigen <- kpca_result@eig

  # Create a data frame with the principal components and the class labels
  data_pca <- data.frame(PC1 = pc1, PC2 = pc2, Cluster = data$cluster)
  #Create a data frame for the skree plot
  scree_df <- data.frame(PC = 1:length(eigen), Eigenvalue = eigen)
  # Create a scatter plot of the principal components, colored by class
  plot <- ggplot(data_pca, aes(x = PC1, y = PC2, color = Cluster)) +
    geom_point() +
    labs(x = "Principal Component 1", y = "Principal Component 2", title = paste("Kernel PCA Results (d
  # Create a scree plot for PCS
  skree_plot <- ggplot(scree_df, aes(x = PC, y = Eigenvalue)) +
    geom_line() +
    geom_point() +
    scale_x_continuous(breaks = 1:length(eigen)) +
    labs(x = "Principal Component", y = "Variance", title = paste("Scree Plot (d =", d, ")")) +
    theme_minimal()
  # Add the plot to the list
  plot_list[[paste("Plt_D", d, sep = "")]] <- plot
  kpca_list <- append(kpca_list,list(data_pca))
  skree_list[[paste("Scree_D", d, sep = "")]] <- skree_plot
}
#Plot PCS  in a grid
ggarrange(plotlist = plot_list, ncol = 2, nrow = 4,common.legend = TRUE)
```



As we can see the odd power kernels give a clearer seperation between the classes, and it appears that higher order terms give higher seperation. However notice that the mini 'ying-yang' shoots of data are shrinking, so it may be that they are just being more compressed to a point, and not necessarily being seperated any better. To deduce whether this is true we will need to train a general linear model on our data. But first lets see the scree plots to find an appropriate number of principal components to use.

```
#Plot the skree plots in a grid
ggarrange(plotlist = skree_list, ncol = 2, nrow = 4,common.legend = TRUE)
```



One thing of note is that we still have a fair amount of variance explained by the 3rd, and 4th components too, so a lack of seperability over PC1&2 is not the end of the world. We choose to limit our general linear model to 4 principal components as it is a good average over all our parameters.

```
# Initialize an empty vector to store the mean squared errors
acc_list <- numeric(length(kpca_list))

# Loop over the data frames in the list
for (i in seq_along(kpca_list)) {
  # Shuffle the data
  shuffled_data <- kpca_list[[i]][sample(nrow(kpca_list[[i]])), ]
  shuffled_data$Cluster <- as.numeric(shuffled_data$Cluster)
  # Calculate the number of rows for the training set (90% of total rows)
  train_rows <- round(0.8 * nrow(shuffled_data))

  # Create the training set and the test set
  trainSet <- head(shuffled_data, n = train_rows)
  testSet <- tail(shuffled_data, n = nrow(shuffled_data) - train_rows)


  # Fit a general linear model on the training set
  model <- glm(Cluster ~ ., data = trainSet)

  # Predict the test set results
  predictions <- round(predict(model, newdata = testSet))
  # Calculate the mean squared error
  acc_list[i] <- mean(testSet$Cluster == predictions)
}
names(acc_list) <- paste0("Degree ", 4:11," kernel")
# Print the mean squared errors
print(acc_list)
```

```
##  Degree 4 kernel  Degree 5 kernel  Degree 6 kernel  Degree 7 kernel
##            0.575            0.585            0.630            0.710
```

```
##  Degree 8 kernel  Degree 9 kernel Degree 10 kernel Degree 11 kernel
##            0.660            0.710             0.700            0.675
```

As we can see, despite the seperation appearing more clear, our linear model doesn't perform very strongly, and in fact it is consistently hovering around 64% no matter the dimension, hinting to diminishing returns. Considering polynomials have left us with nothing, we move on to an infinite-dimensional kernel. Fun fact, I initially trained my general linear model on the first 2 principal components, and the results were way better, we're talking 20% better.

Gaussian Kernel

Recall that the gaussian kernel is as follows:

$$k(x, x') = \exp\left(-\frac{||x - x'||^2}{\gamma}\right) \text{ for } \gamma > 0$$

Gaussian kernel are an all purpose useful kernel, hopefully they can help us tackle the nightmare that are our concentric ball data. We build a function below which runs a gaussian kernel PCA on the dataset for a given free parameter γ.

```r
library(kernlab)
library(ggplot2)
set.seed(123)

run_gaussian_kernel_pca <- function(data, gamma) {
  # Perform kernel PCA with a Gaussian kernel
  kpca_result <- kpca(~., data = data, kernel = "rbfdot", kpar = list(sigma = gamma),features = 0)

  # Extract the principal components
  pc1 <- kpca_result@rotated[, 1]
  pc2 <- kpca_result@rotated[, 2]
  pc3 <- kpca_result@rotated[, 3]
  pc4 <- kpca_result@rotated[, 4]

  # Create a data frame with the principal components and the class labels
  data_pca <- data.frame(PC1 = pc1, PC2 = pc2, PC3 = pc3, PC4=pc4, Cluster = data$cluster)

  # Extract eigenvalues
  eigenvalues <- kpca_result@eig

  # Create a data frame for the scree plot
  scree_df <- data.frame(PC = 1:length(eigenvalues), Eigenvalue = eigenvalues)

  # Create a scree plot
  scree_plot <- ggplot(scree_df, aes(x = PC, y = Eigenvalue)) +
    geom_line() +
    geom_point() +
    scale_x_continuous(breaks = 1:length(eigenvalues)) +
    labs(x = "Principal Component", y = "Variance", title = paste("Scree Plot (gamma =", gamma, ")")) +
    theme_minimal()

  # Create a scatter plot of the principal components, colored by class
  pca_plot <- ggplot(data_pca, aes(x = PC1, y = PC2, color = Cluster)) +
```

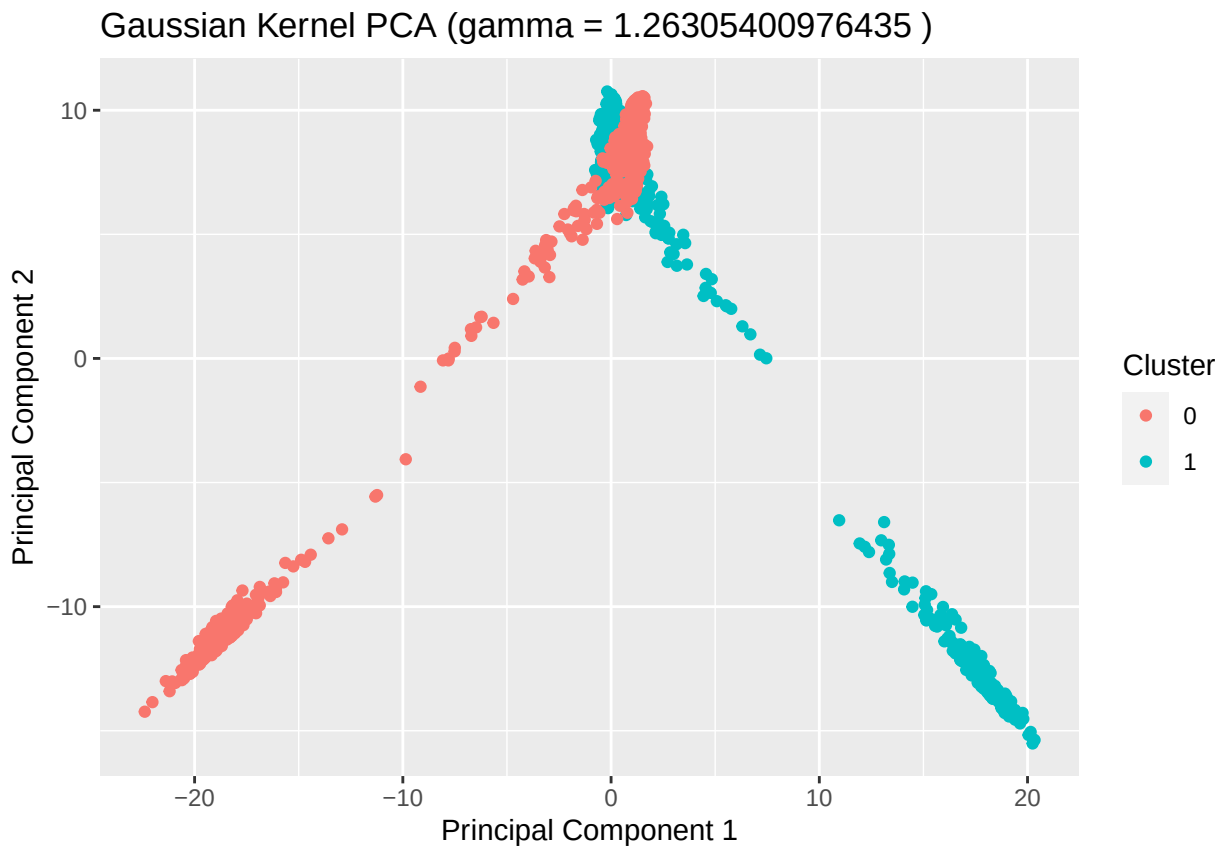
```
    geom_point() +
    labs(x = "Principal Component 1", y = "Principal Component 2", title = paste("Gaussian Kernel PCA (
  # Create a scatter plot of PC3 and PC4
  pca_plot2 <- ggplot(data_pca, aes(x = PC3, y = PC4, color = Cluster)) +
    geom_point() +
    labs(x = "Principal Component 1", y = "Principal Component 2", title = paste("Gaussian Kernel PCA (

  return(list(data_pca = data_pca, pca_plot = pca_plot, scree_plot = scree_plot, pca_plot2 = pca_plot2)
}
```

We have a large range of γ values to choose from, to get a good idea we will incorporate the *Median Trick*, which is to take the median of the pairwise distances of the data as the γ value, i.e:

$$\gamma = \frac{1}{2}\text{median}\left\{||x_i - x_j||^2\right\}.$$

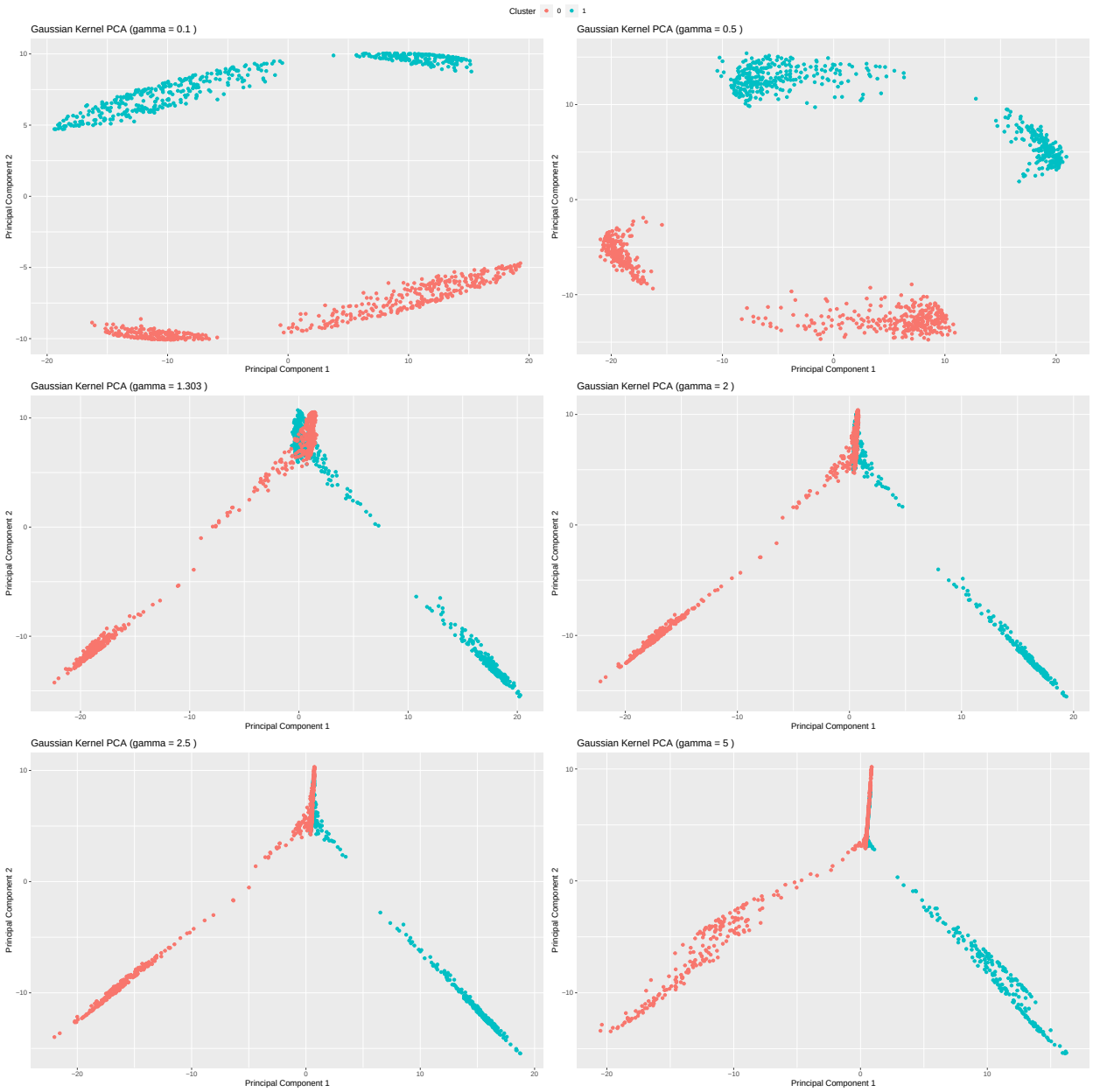
Let's formulate this in R and run our gaussian kernel PCA.

```
# Calculate the median of the pairwise distances for a good gamma
pairwise_distances <- as.matrix(dist(data[,1:3]))
median_gamma <- 1/2*median(pairwise_distances^2)

# Run Gaussian kernel PCA with the median gamma value
gaussian_kpca_result <- run_gaussian_kernel_pca(data, median_gamma)
gaussian_kpca_result$pca_plot
```



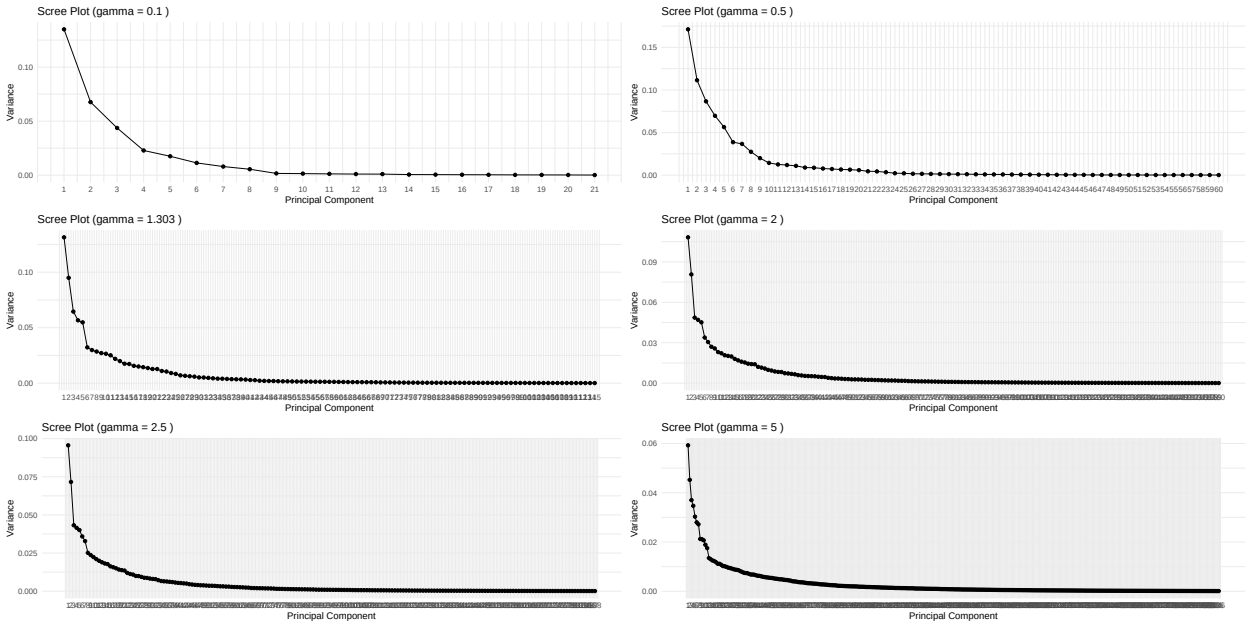
A familiar result to be sure, very similar to the polynomial kernel. Unfortunately we expect that a general linear model would not be sufficient in this case, at least when restricted to only 2 components. Let's try a few more γ values to see if we can get a better result.

```
set.seed(123)
# Run Gaussian kernel PCA with a few different gamma values
gamma_values <- c(0.1, 0.5,1.303, 2.0, 2.5, 5)
#Create a list to store the plots
plot_list <- list()
skree_list <- list()
data_pca_list <- list()
for (gamma in gamma_values) {
  # Run Gaussian kernel PCA
  gaussian_kpca_result <- run_gaussian_kernel_pca(data, gamma)

  # Add the PCA plot to the list
  plot_list[[paste("Plt_Gamma", gamma, sep = "")]] <- gaussian_kpca_result$pca_plot
  skree_list[[paste("Scree_Gamma", gamma, sep = "")]] <- gaussian_kpca_result$scree_plot
  data_pca_list[[paste("Data_Gamma", gamma, sep = "")]] <- gaussian_kpca_result$data_pca
}
#Plot the PCA plots in a grid
ggarrange(plotlist = plot_list, ncol = 2, nrow = 3,common.legend = TRUE)
```



Wow! That's looking amazing, we have very distinct seperation that could be perfectly fit with a general linear model, of course we are only viewing 2 principle components, but it's a good sign. Let's see the skree plots for these γ values.

```
#Plot the skree plots in a grid
ggarrange(plotlist = skree_list, ncol = 2, nrow = 3,common.legend = TRUE)
```



In our best case seperation we see that the first 4 principal components account for the majority of the variance, after that any variance is below 5%. So we fit a general linear model on the first four principal components, as we did in the polynomial case:

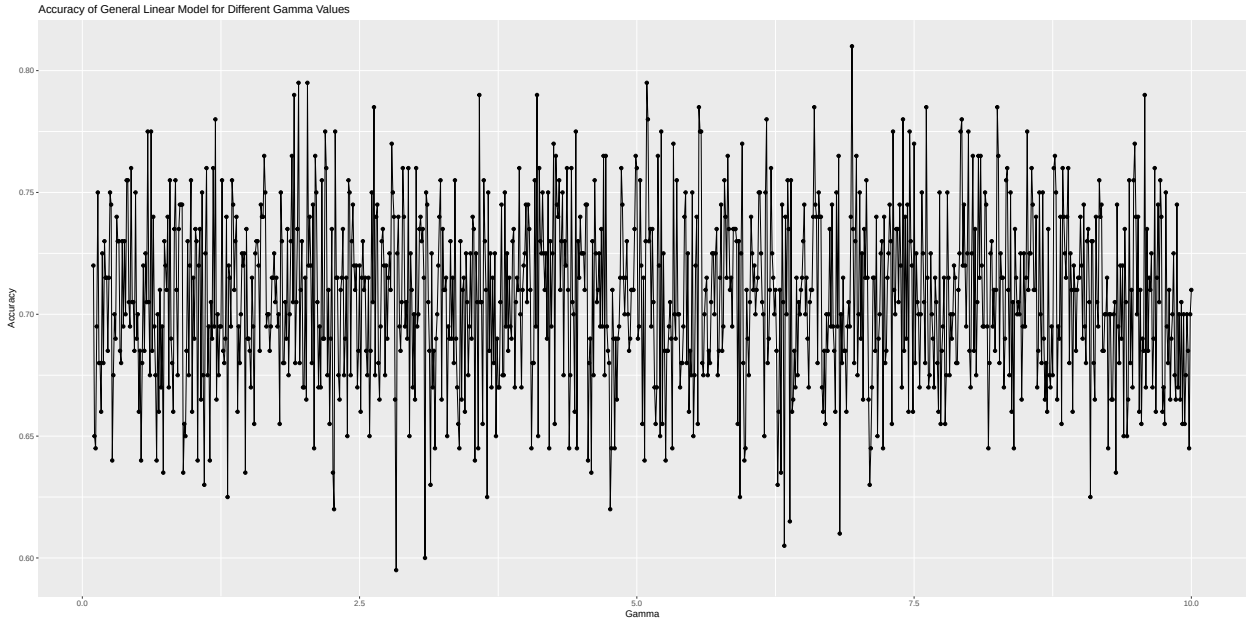
```
set.seed(123)
#Create function which verifies accuracy of rbf kernel pca classifier
gamma_values <- c(0.1, 0.5, 1.303, 2.0, 2.5, 5)

calculate_accuracy <- function(gamma_values) {
  # Initialize an empty vector to store the accuracies
  acc_list <- numeric(length(gamma_values))

  # Loop over the gamma values
  for (i in seq_along(gamma_values)) {
    # Shuffle the data
    shuffled_data <- gaussian_kpca_result$data_pca[sample(nrow(gaussian_kpca_result$data_pca)), ]
    shuffled_data$Cluster <- as.numeric(shuffled_data$Cluster)
    # Calculate the number of rows for the training set (90% of total rows)
    train_rows <- round(0.8 * nrow(shuffled_data))

    # Create the training set and the test set
    trainSet <- head(shuffled_data, n = train_rows)
    testSet <- tail(shuffled_data, n = nrow(shuffled_data) - train_rows)

    # Fit a general linear model on the training set
    model <- glm(Cluster ~ ., data = trainSet)

    # Predict the test set results
    predictions <- round(predict(model, newdata = testSet))
    # Calculate the accuracy
    acc_list[i] <- mean(testSet$Cluster == predictions)
  }

  # Assign names to the accuracy values
```

```
  names(acc_list) <- paste0("Gamma ", gamma_values, " kernel")

  # Return the accuracy list
  return(acc_list)
}

# Call the function with gamma_values
accuracy_results <- calculate_accuracy(gamma_values)

# Print the accuracy results
print(accuracy_results)
```

```
##   Gamma 0.1 kernel   Gamma 0.5 kernel Gamma 1.303 kernel     Gamma 2 kernel
##              0.720              0.650              0.645              0.695
##   Gamma 2.5 kernel     Gamma 5 kernel
##              0.750              0.680
```

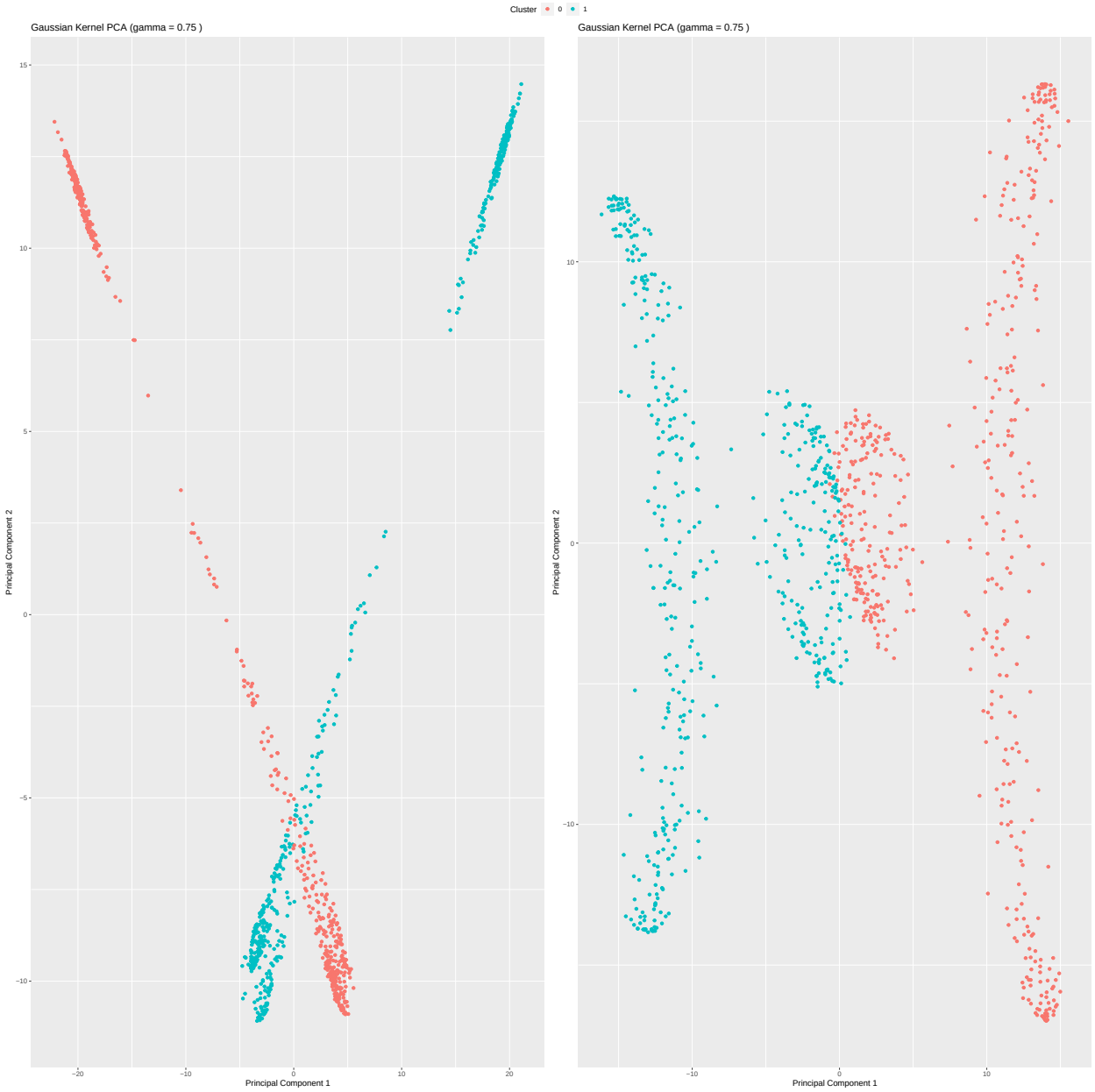
This is a slight improvement over the polynomial kernel, but still not great. Surprisingly $\gamma = 0.2$ performed best with 70% and the median trick did *fine*, but 70% accuracy won't do if we're trying to detect active vs inactive landmines. On that paranoid note, lets put some more effort in, below we vary over a large range for γ and plot the accuracy, maybe we'll even plot the PCs as a bonus if it goes well!

```
set.seed(123)

# Create a sequence of gamma values
gamma_values <- seq(0.1, 10, by = 0.01)

# Calculate the accuracy for each gamma value
acc_list <- calculate_accuracy(gamma_values)

# Plot the accuracy
accuracy_plot <- ggplot(data.frame(Gamma = gamma_values, Accuracy = acc_list), aes(x = Gamma, y = Accura
  geom_line() +
  geom_point() +
  labs(x = "Gamma", y = "Accuracy", title = "Accuracy of General Linear Model for Different Gamma Value

accuracy_plot
```



As expected this is a very chaotic system, but we do have a pretty substainial peak, at $\gamma = 0.75$ with 82.5% accuracy. Let's see what the principal components look like.

```
# Run Gaussian kernel PCA with the best gamma value
best_gamma <- 0.75
best_gaussian_kpca_result <- run_gaussian_kernel_pca(data, best_gamma)
ggarrange(best_gaussian_kpca_result$pca_plot,best_gaussian_kpca_result$pca_plot2,ncol = 2, common.legend
```



Now this is some cool seperation, despite PC1 and PC2 not being particularly seperable, we have extremely distinct seperation in PC3 and PC4. Obviously we know from before that this is 82.5% effective. In summary RBF kernel is consistently reliable for classification via kernel PCA method, and far outmatches a polynomial kernel, which may have been the 'smart' choice given the structure of the data, maybe a trig kernel could be worth exploring next. In the end we acheived 82.5% accuracy, which I guess still wouldn't be great withlandmines. Fortunately there aren't many landmines in Rstudio.