portfolio_3

Henry Bourne

2023-02-06

Kernel Principal Component Analysis

First we will generate our training dataset that we will use for this task and visualize it:

```
library(Rfast)
```

```
## Loading required package: Rcpp
```

```
## Loading required package: RcppZiggurat
```

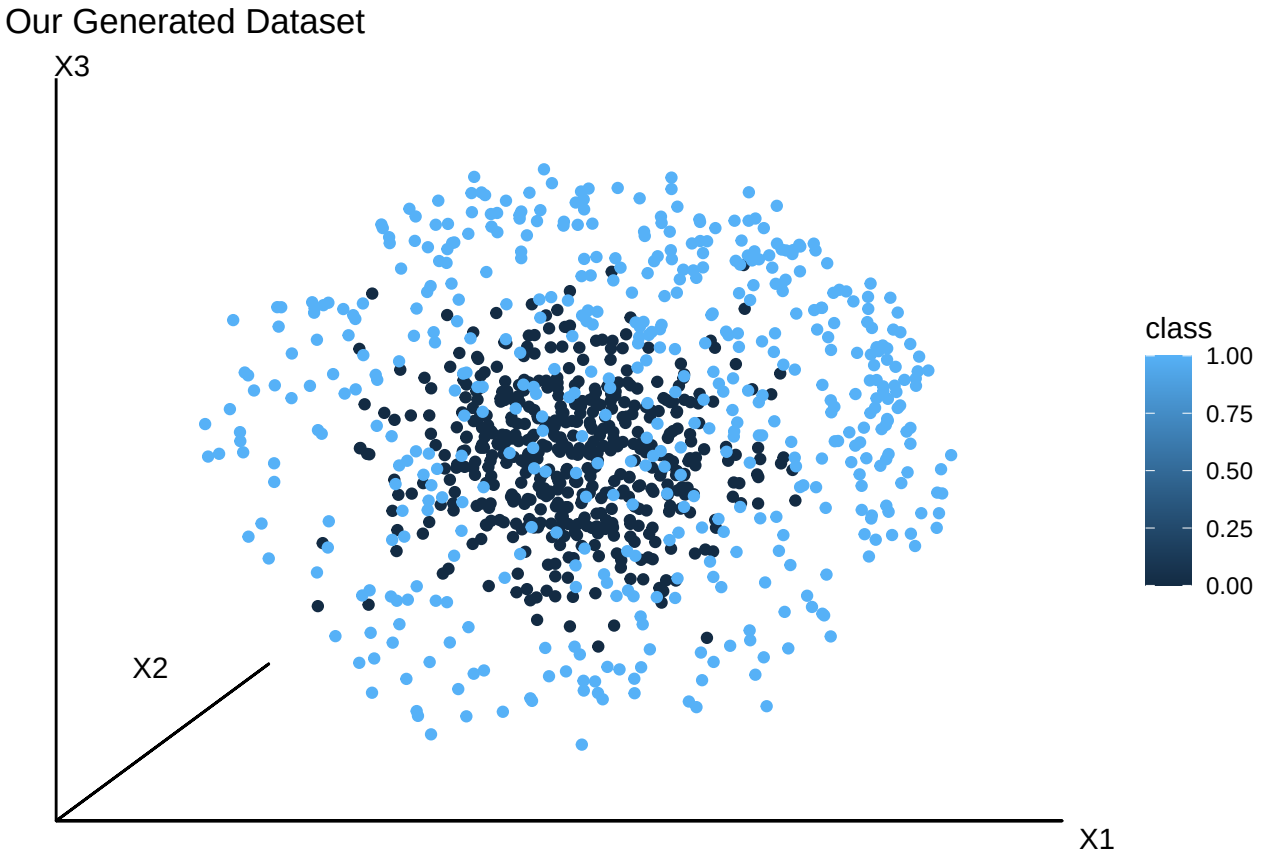
```
n <- 1000
d <- 3
C0 <- rmvnorm(n/2, rep(0,d), diag(rep(0.07, d)))
C1 <- rvmf(n/2, mu = c(1, 1, 1) / sqrt(10), k = 1) + rmvnorm(n/2, rep(0,d), diag(rep(0.005, d)))
D <- data.frame(rbind(C0, C1), "class"= c(rep(0,n/2), rep(1,n/2) ) )

library("gg3D")
```

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

```
ggplot(D, aes(x=X1, y=X2, z=X3, color=class)) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
            angle=c(0,0,0),
            hjust=c(0,2,2),
            vjust=c(2,2,-2)) +
  ggtitle("Our Generated Dataset")
```

```
## Warning: The following aesthetics were dropped during statistical transformation: z,
## colour
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
## The following aesthetics were dropped during statistical transformation: z,
## colour
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```



As you can see this dataset is clearly not linearly separable, however, it is indeed separable. We will use this dataset in our task to perform classification using PCA and KPCA and perform an analysis on classification performance. Let's now generate the test dataset which we will use later to evaluate the performance of our trained classifiers:

```
n.test <- 200
C0.test <- rmvnorm(n.test/2, rep(0,d), diag(rep(0.07, d)))
C1.test <- rvmf(n.test/2, mu = c(1, 1, 1) / sqrt(10), k = 1) + rmvnorm(n.test/2, rep(0,d), diag(rep(0.0
Test <- data.frame(rbind(C0.test, C1.test), "class"= c(rep(0,n.test/2), rep(1,n.test/2) ) )
```

PCA vs KPCA

In this subsection we will perform binary classification using data dimensionality reduction performed by PCA and KPCA. We will then compare their performance.

Let's now move onto performing our principal component analyses. First lets carry out PCA on the data matrix, note that we won't scale or center the data as its already centered and scaled. By not scaling we get the bonus of preserving variance in our transformation:

```
pc <- prcomp(~ . -class, D, scale. = FALSE, retx=TRUE)
pc
```

```
## Standard deviations (1, .., p=3):
## [1] 0.4539823 0.4506204 0.4284990
##
## Rotation (n x k) = (3 x 3):
##            PC1        PC2        PC3
## X1  0.67432201 -0.2180488 -0.7055101
```

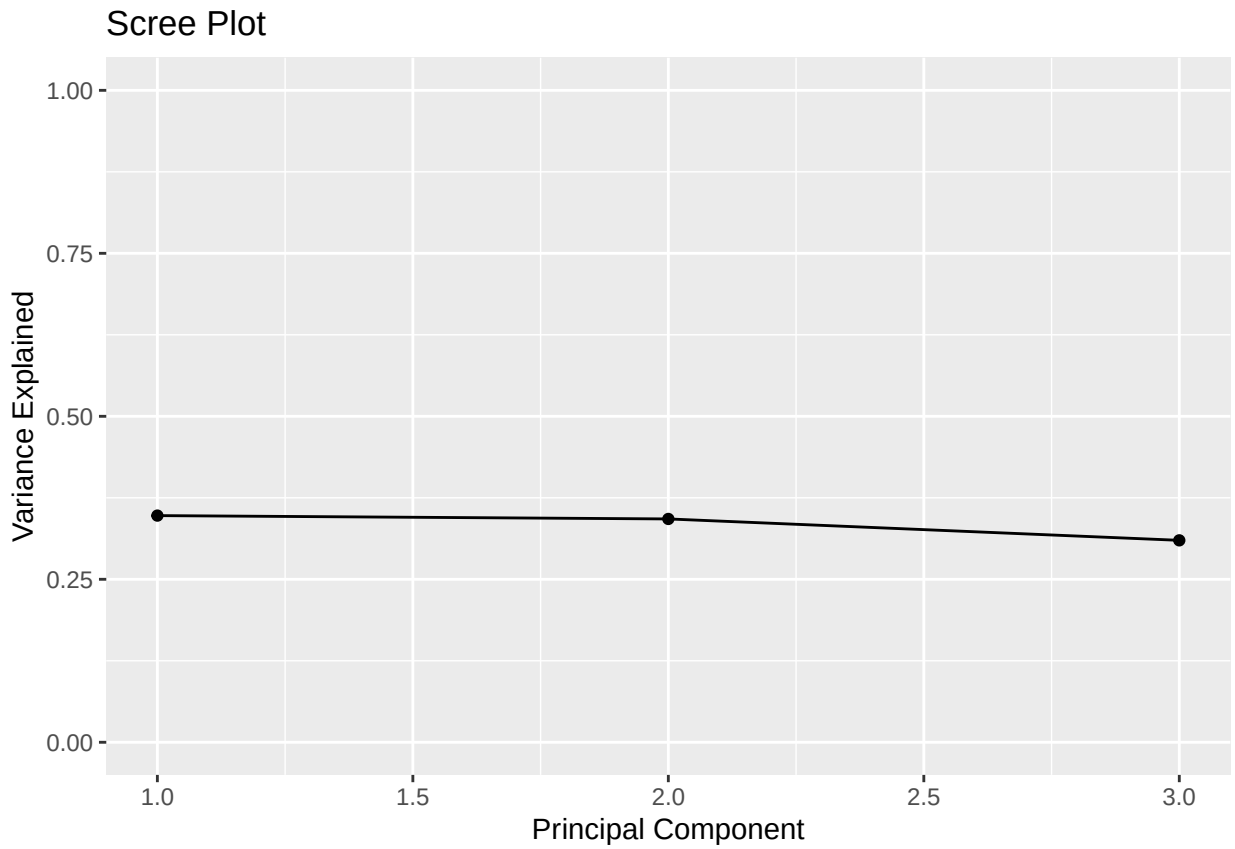
```
## X2 -0.07682201  0.9295106 -0.3607055
## X3  0.73443054  0.2974304  0.6100385
```

Let's produce a scree plot and select the number of principal components we want to use:

```
var_prcnt = pc$sdev^2 / sum(pc$sdev^2)

qplot(c(1:3), var_prcnt) +
  geom_line() +
  xlab("Principal Component") +
  ylab("Variance Explained") +
  ggtitle("Scree Plot") +
  ylim(0, 1)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```



We see that the variance accounted for by each principal component is roughly the same, which makes sense as looking at the plot above the variance is roughly the same in every direction. Hence, we will keep all three principal components for the rest of the analysis.

Now let's perform KPCA, using the radial basis kernel function and again we will not scale or center the data:

```
library(kernlab)
```

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

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

```
kpc.0 <- kpca(~ . -class, D, kernel="rbfdot")
head(pcv(kpc.0))
```

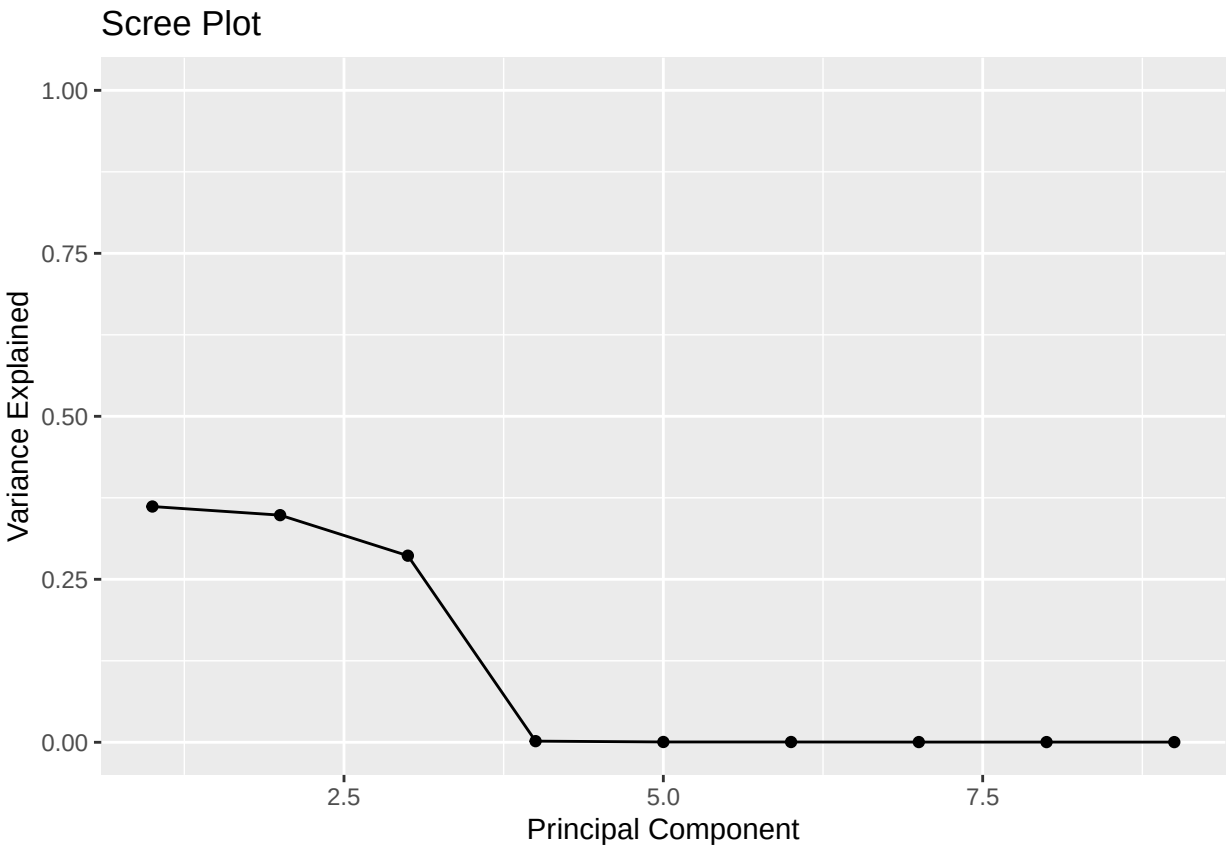
```
##              [,1]        [,2]         [,3]       [,4]         [,5]        [,6]
## [1,] -0.005815906 -0.25190447  0.093685699 -0.1935810 -0.487789559 -0.43624868
## [2,]  0.041030751 -0.11132942 -0.116859899 -0.6928178  0.074528089 -0.41853248
## [3,] -0.051729482  0.07179196  0.080158157 -0.7977406  0.001052557 -0.18043949
## [4,] -0.019504492 -0.13393336 -0.018869806 -0.7403287 -0.181009740 -0.25266867
## [5,] -0.039283631  0.03488526 -0.008843167 -0.8982144 -0.053778435 -0.02978801
## [6,]  0.004626681 -0.25320655 -0.016502270 -0.2528753 -0.199820499 -0.82455356
##              [,7]        [,8]        [,9]
## [1,]  0.818401328  0.19214809 -0.82242319
## [2,] -0.055183020 -0.14722040  0.16918086
## [3,] -0.004294162 -0.04704144  0.13619144
## [4,]  0.203203136 -0.27117506 -0.10211887
## [5,]  0.066935254 -0.07785086 -0.06916394
## [6,]  0.764820963 -0.36195751 -0.37421077
```

```
# We only print first 6 rows
```

Let's again create a scree plot in order to help us select the number of principal components we would like to keep:

```
var_prcnt = eig(kpc.0)^2 / sum(eig(kpc.0)^2)

qplot(c(1:length(eig(kpc.0))), var_prcnt) +
  geom_line() +
  xlab("Principal Component") +
  ylab("Variance Explained") +
  ggtitle("Scree Plot") +
  ylim(0, 1)
```



From the scree plot it's very clear that the first three principal components contain nearly all the variance, hence, we will use only the first three components. Let's update our KPCA to reflect this choice:

```
kpc.0 <- kpca(~ . -class, data=D, kernel="rbfdot", features=3)
```

Let's now train a logistic model on the transformed data from our PCA

```
D.rt <- data.frame(pc$x, "class"=D[,4])
model.pc <- glm(class~ ., family=binomial(link='logit'), data=D.rt)
model.pc
```

```
##
## Call:  glm(formula = class ~ ., family = binomial(link = "logit"), data = D.rt)
##
## Coefficients:
## (Intercept)          PC1          PC2          PC3
##   0.0008955    1.3231664    0.9224770   -0.3849362
##
## Degrees of Freedom: 999 Total (i.e. Null);  996 Residual
## Null Deviance:       1386
## Residual Deviance: 1267  AIC: 1275
```

And now on our transformed data from our KPCA:

```
D.rt <- data.frame(rotated(kpc.0), "class"=D[,4])
model.kpc.0 <- glm(class~ ., family=binomial(link='logit'), data=D.rt)
model.kpc.0
```

```
##
```

```
## Call:  glm(formula = class ~ ., family = binomial(link = "logit"), data = D.rt)
##
## Coefficients:
## (Intercept)           X1           X2           X3
##    0.002903     0.121999    -0.049810    -0.030357
##
## Degrees of Freedom: 999 Total (i.e. Null);  996 Residual
## Null Deviance:       1386
## Residual Deviance: 1253  AIC: 1261
```

Let's now test our models' performance on the testing data:

```
# Let's first obtain our predictions for the test data and calculate the accuracy for PCA
Test.rt <- data.frame(as.matrix(Test[,-4]) %*% pc$rotation, "class" = Test[,4])
preds.pc <- predict(model.pc, Test.rt, type="response")
preds.pc <- ifelse(preds.pc > 0.5,1,0)
acc.pc <- 1-mean(preds.pc != Test[,4])

# Let's now obtain our predictions for the test data and calculate the accuracy for KPCA
Test.rt <- predict(kpc.0,Test)
Test.rt <- data.frame("X1" = Test.rt[,1], "X2" = Test.rt[,2], "X3" = Test.rt[,3], "class" = Test[,4])
preds.kpc.0 <- predict(model.kpc.0, Test.rt, type="response")
preds.kpc.0 <- ifelse(preds.kpc.0 > 0.5,1,0)
acc.kpc.0 <- 1-mean(preds.kpc.0 != Test[,4])


# Let's now print the accuracies we obtained
acc.pc # accuracy using PCA
```

```
## [1] 0.63
```

```
acc.kpc.0 # accuracy using KPCA
```

```
## [1] 0.7
```

We see that the classification accuracy obtained on the test set by performing binary logistic regression on the dimensionality reduction given by KPCA is much higher than that given by PCA. Note, however, that we have applied KPCA here very naively with no investigation into which kernel we should use and with which hyperparameters. In this next section we will carry out this investigation.

Tuning Hyperparameters

We will now compare classification performance of fitting a logistic regression on transformed data from KPCA where we perform KPCA using different kernels and hyper-parameters. In particular there are 7 kernel functions we will investigate:

- Radial basis kernel function
- Laplacian kernel function
- Linear kernel function
- Polynomial kernel function

Each of these also have their own hyper-parameters. What we will do is first investigate the optimal choice of hyper-parameters and features to be kept for each kernel and then compare the performance of the kernels using their optimal hyperparameters. We will start by choosing optimal hyper-parameters for the rbf kernel, we will select the step size by calculating the kaiser criterion:

```r
# We calculate the median heuristic
median_heuristic <- medianTune(as.matrix(D[,-4])); median_heuristic
```

```
## [1] 0.7067554
```

```r
# We create list of hyperparameters
sigma <- c(0.01, 0.1, 0.5, 1, 2, 3) * median_heuristic

# We perform KPCA using rbf kernel with each hyper-parameter,
# then calculate and store their performance on the test set after fitting a logistic regression
accs.rbf <- rep(NA, length(sigma))
names(accs.rbf) = paste0( "Sigma=" , paste(signif(sigma, 4)))
for(i in 1:length(sigma)){
  # We initialize list with hyper-parameters for this iteration
  kpar = list(sigma=sigma[i])

  # We do KPCA
  kpc.rbf <- kpca(~ . -class, D, kernel="rbfdot", kpar=kpar)

  # We calc Keiser criterion
  k <- max(which(eig(kpc.rbf) > sum(eig(kpc.rbf))/length(eig(kpc.rbf))))

  # We narrow down features according to Keiser criterion
  kpc.rbf <- kpca(~ . -class, D, kernel="rbfdot", kpar=kpar, features = k)

  # We carry out logistic regression
  D.rt <- data.frame(rotated(kpc.rbf), "class"=D[,4])
  model.kpc.rbf <- glm(class~ ., family=binomial(link='logit'), data=D.rt)

  # We perform testing
  Test.rt <- predict(kpc.rbf,Test)
  Test.rt <- data.frame(Test.rt, "class" = Test[,4])
  colnames(Test.rt) <- colnames(D.rt)
  preds.kpc.rbf <- predict(model.kpc.rbf, Test.rt, type="response")
  preds.kpc.rbf <- ifelse(preds.kpc.rbf > 0.5,1,0)

  # We calculate accuracy of testing
  accs.rbf[i] <- 1-mean(preds.kpc.rbf != Test[,4])
}
accs.rbf
```

```
## Sigma=0.007068  Sigma=0.07068   Sigma=0.3534   Sigma=0.7068    Sigma=1.414
##          0.700          0.700          0.985          0.975          0.975
##     Sigma=2.12
##          0.975
```

We see that the largest accuracy on the test set is obtained when the bandwidth sigma=0.704 (4 s.f.), which is the value of the median heuristic. So we will take this to be the best value for the bandwidth when using the RBF kernel.

Now we will do the same but for the

```r
# We create list of hyperparameters
sigma <- c(0.01, 0.1, 0.5, 1, 2, 3, 5) * median_heuristic

# We perform KPCA using laplacian kernel with each hyper-parameter,
```

```
# then calculate and store their performance on the test set after fitting a logistic regression
accs.laplace <- rep(NA, length(sigma))
names(accs.laplace) = paste0( "Sigma=" , paste(signif(sigma, 4)))
for(i in 1:length(sigma)){
  # We initialize list with hyper-parameters for this iteration
  kpar = list(sigma=sigma[i])

  # We do KPCA
  kpc.laplace <- kpca(~ . -class, D, kernel="laplacedot", kpar=kpar)

  # We calc Keiser criterion
  k <- max(which(eig(kpc.laplace) > sum(eig(kpc.laplace))/length(eig(kpc.laplace))))

  # We narrow down features according to Kaiser criterion
  kpc.laplace <- kpca(~ . -class, D, kernel="laplacedot", kpar=kpar, features = k)

  # We carry out logistic regression
  D.rt <- data.frame(rotated(kpc.laplace), "class"=D[,4])
  model.kpc.laplace <- glm(class~ ., family=binomial(link='logit'), data=D.rt)

  # We perform testing
  Test.rt <- predict(kpc.laplace,Test)
  Test.rt <- data.frame(Test.rt, "class" = Test[,4])
  colnames(Test.rt) <- colnames(D.rt)
  preds.kpc.laplace <- predict(model.kpc.laplace, Test.rt, type="response")
  preds.kpc.laplace <- ifelse(preds.kpc.laplace > 0.5,1,0)

  # We calculate accuracy of testing
  accs.laplace[i] <- 1-mean(preds.kpc.laplace != Test[,4])
}
accs.laplace
```

```
## Sigma=0.007068  Sigma=0.07068   Sigma=0.3534   Sigma=0.7068    Sigma=1.414
##          0.720          0.975          0.975          0.980          0.970
##     Sigma=2.12    Sigma=3.534
##          0.945          0.960
```

Tuning the bandwidth for the Laplace kernel gives us slightly different results with sigma= 2.112 giving the best results, which is different to the sigma given by the median heuristic.

Let's now tune the polynomial kernel:

```
# We create list of hyper-parameters
degree <- c(1,2,4,6,8,10)

# We perform KPCA using polynomial kernel with each hyper-parameter,
# then calculate and store their performance on the test set after fitting a logistic regression
accs.poly <- rep(NA, length(degree))
names(accs.poly) = paste0( "Degree=" , paste(signif(degree, 4)))
for(i in 1:length(degree)){
  # We initialize list with hyper-parameters for this iteration
  kpar = list(degree=degree[i])

  # We do KPCA
  kpc.poly <- kpca(~ . -class, D, kernel="polydot", kpar=kpar)
```

```
  # We calc Keiser criterion
  k <- max(which(eig(kpc.poly) > sum(eig(kpc.poly))/length(eig(kpc.poly))))

  # We narrow down features according to Kaiser criterion
  kpc.poly <- kpca(~ . -class, D, kernel="polydot", kpar=kpar, features = k)

  # We carry out logistic regression
  D.rt <- data.frame(rotated(kpc.poly), "class"=D[,4])
  model.kpc.poly <- glm(class~ ., family=binomial(link='logit'), data=D.rt)

  # We perform testing
  Test.rt <- predict(kpc.poly,Test)
  Test.rt <- data.frame(Test.rt, "class" = Test[,4])
  colnames(Test.rt) <- colnames(D.rt)
  preds.kpc.poly <- predict(model.kpc.poly, Test.rt, type="response")
  preds.kpc.poly <- ifelse(preds.kpc.poly > 0.5,1,0)

  # We calculate accuracy of testing
  accs.poly[i] <- 1-mean(preds.kpc.poly != Test[,4])
}
accs.poly
```

```
##  Degree=1  Degree=2  Degree=4  Degree=6  Degree=8 Degree=10
##     0.700     0.710     0.970     0.970     0.970     0.955
```

We see that the optimal degree to use when working with the polynomial kernel is 4.

Finally we would also like to compare the above kernels with the linear kernel, however, the linear kernel has no hyperparameters to tune so we will move directly onto the analysis.

Comparing kernels

Let's now use out optimal hyper-parameters for each of the kernels and then compare the performance of logistic regression on the resulting dimensionality reduction given by KPCA.

```
kernels <- c("rbfdot", "laplacedot", "polydot", "vanilladot")
hyper_params <- c(median_heuristic, 3* median_heuristic, 4, NA)
n_c <- rep(NA, length(kernels))
accs.kernels <- rep(NA, length(kernels))
names(accs.kernels) = paste0( "kernel=" , kernels)
names(n_c) = paste0( "kernel=" , kernels)
for(i in 1:length(kernels)){
  # We initialize list with hyper-parameters for this iteration
  if((kernels[i] == "rbfdot") | (kernels[i] == "laplacedot") ){
    kpar = list(sigma=hyper_params[i])
  }
  else if(kernels[i] == "polydot"){
    kpar = list(degree=hyper_params[i])
  }
  else{
    kpar = list()
  }

  # We do KPCA
  kpc <- kpca(~ . -class, D, kernel=kernels[i], kpar=kpar)
```

```
  # We calc Keiser criterion
  k <- max(which(eig(kpc) > sum(eig(kpc))/length(eig(kpc))))
  n_c[i] <- k

  # We narrow doum features according to Kaiser criterion
  kpc <- kpca(~ . -class, D, kernel=kernels[i], kpar=kpar, features = k)

  # We carry out logistic regression
  D.rt <- data.frame(rotated(kpc), "class"=D[,4])
  model.kpc <- glm(class~ ., family=binomial(link='logit'), data=D.rt)

  # We perform testing
  Test.rt <- predict(kpc,Test)
  Test.rt <- data.frame(Test.rt, "class" = Test[,4])
  colnames(Test.rt) <- colnames(D.rt)
  preds.kpc <- predict(model.kpc, Test.rt, type="response")
  preds.kpc <- ifelse(preds.kpc > 0.5,1,0)

  # We calculate accuracy of testing
  accs.kernels[i] <- 1-mean(preds.kpc != Test[,4])
}
n_c
```

```
##     kernel=rbfdot kernel=laplacedot    kernel=polydot kernel=vanilladot
##                 9               106                 9                 2
```

```
accs.kernels
```

```
##     kernel=rbfdot kernel=laplacedot    kernel=polydot kernel=vanilladot
##             0.975             0.945             0.970             0.700
```

Our results tell us that the laplacian kernel achieves the highest classification accuracy and also uses the largest number of principal components (105), close in performance are the RBF kernel and polynomial kernel both achieving an accuracy 0.01 below that achieved by the laplacian kernel. However, note the number of features needed by the RBF kernel and polynomial kernel are substantially lower at only 9. Using the linear kernel gave the worst performance by far, note also that the linear kernel is akin to carrying out regular PCA. So by using a (non-linear) kernel we are able to model variance non-linearly and are hence able to get a feature space where the two classes in our dataset are linearly seperable.