# Kernel PCA

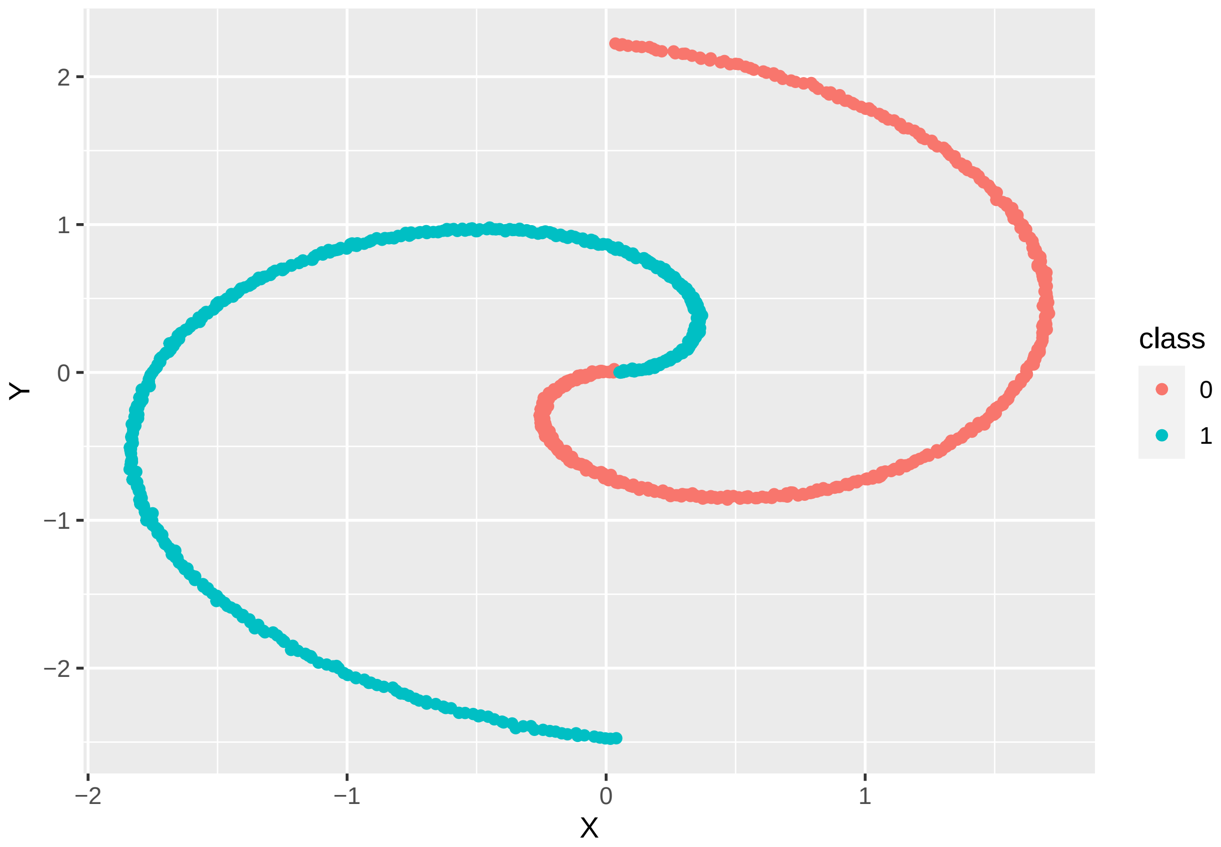
Sam Bowyer

2023-02-10

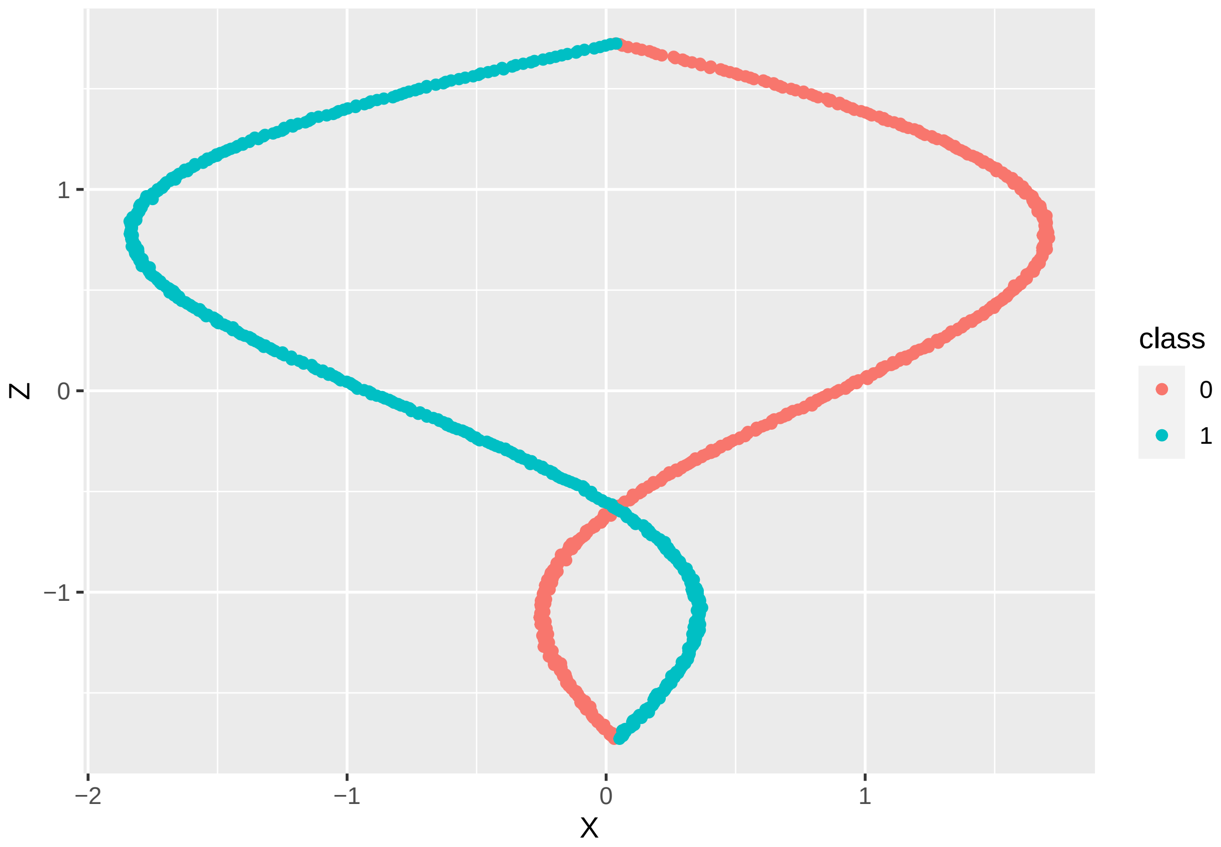
## Generating The Dataset

For this portfolio we want to consider a dataset for classification that isn't linearly separable, as this will (hopefully) allow us to showcase the strengths of kernel PCA compared to regular PCA. In particular, we'll be working with three-dimensional data in the shape of concentric spirals in the $x$-$y$ plane moving along the $z$-axis, generated below.

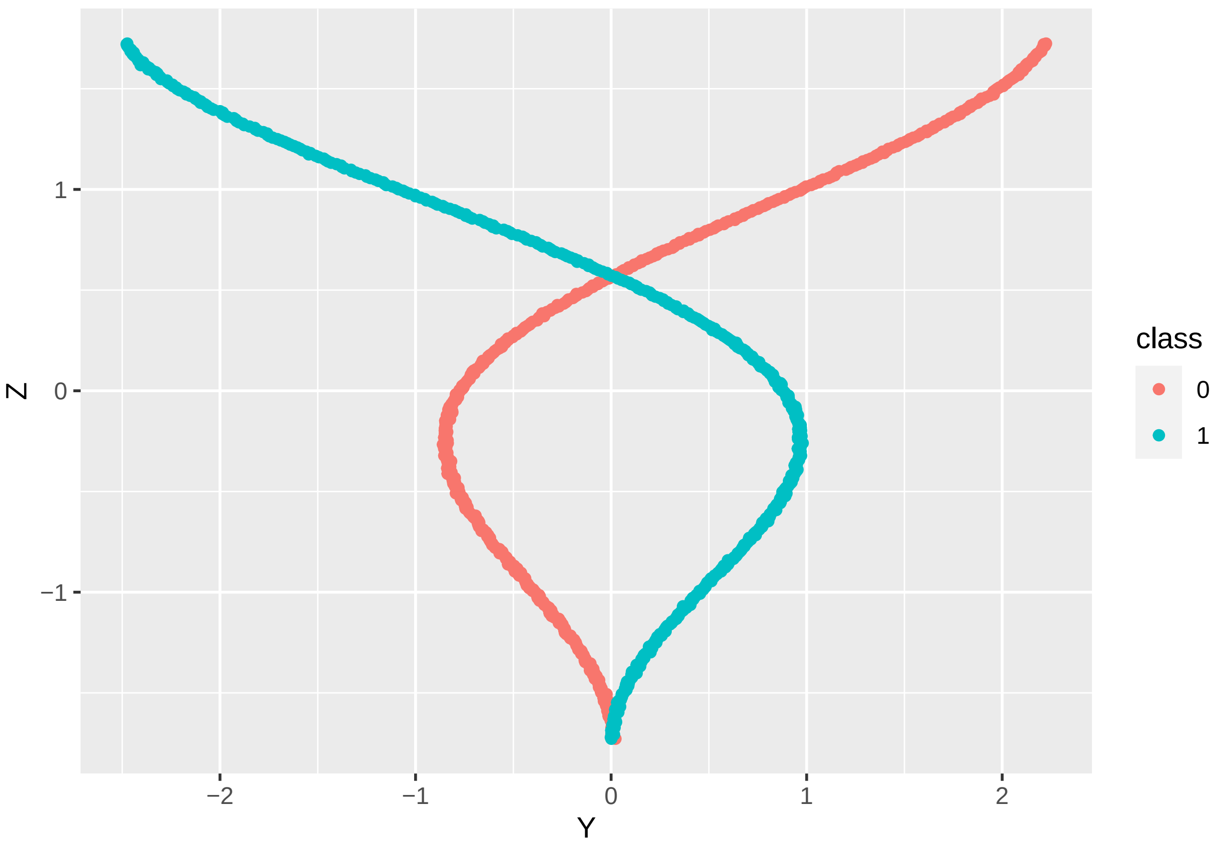
```r
n = 1000  # Total number of data points

Xs = matrix(rep(0,n*3), nrow=n)

for (i in 1:n/2){
  for (class in 0:1){
    coords = c(cos(i*3*pi/(n)), sin(i*3*pi/(n))) * (class*2 -1) * ((i*(8+class))/6)
    coords = coords + rnorm(2, 0, c(2,2))    # Add some noise
    Xs[class*n/2 + i,] = c(coords, i)  # add the z=i dimension to moves spirals along z axis
  }
}
Xs = scale(Xs)

# Put data into a dataframe
data = as.data.frame(cbind(Xs, c(rep(0,n/2), rep(1,n/2))))
colnames(data) = c("X", "Y", "Z", "class")
data[,"class"] = as.factor(data[,"class"])

# Plot
library(ggplot2)
ggplot(data = data, aes(X, Y, color = class)) +
  geom_point()
```



```
ggplot(data = data, aes(X, Z, color = class)) +
  geom_point()
```



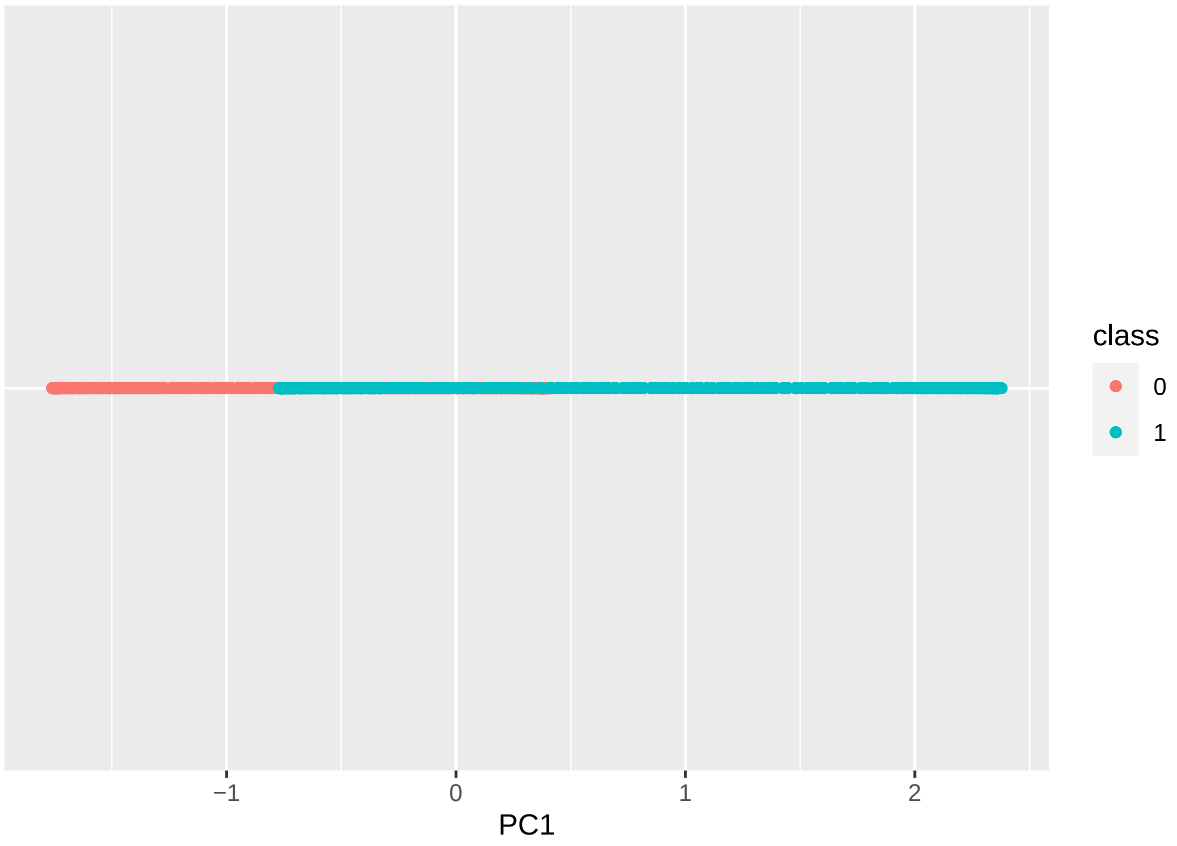
```
ggplot(data = data, aes(Y, Z, color = class)) +
  geom_point()
```



## Regular PCA

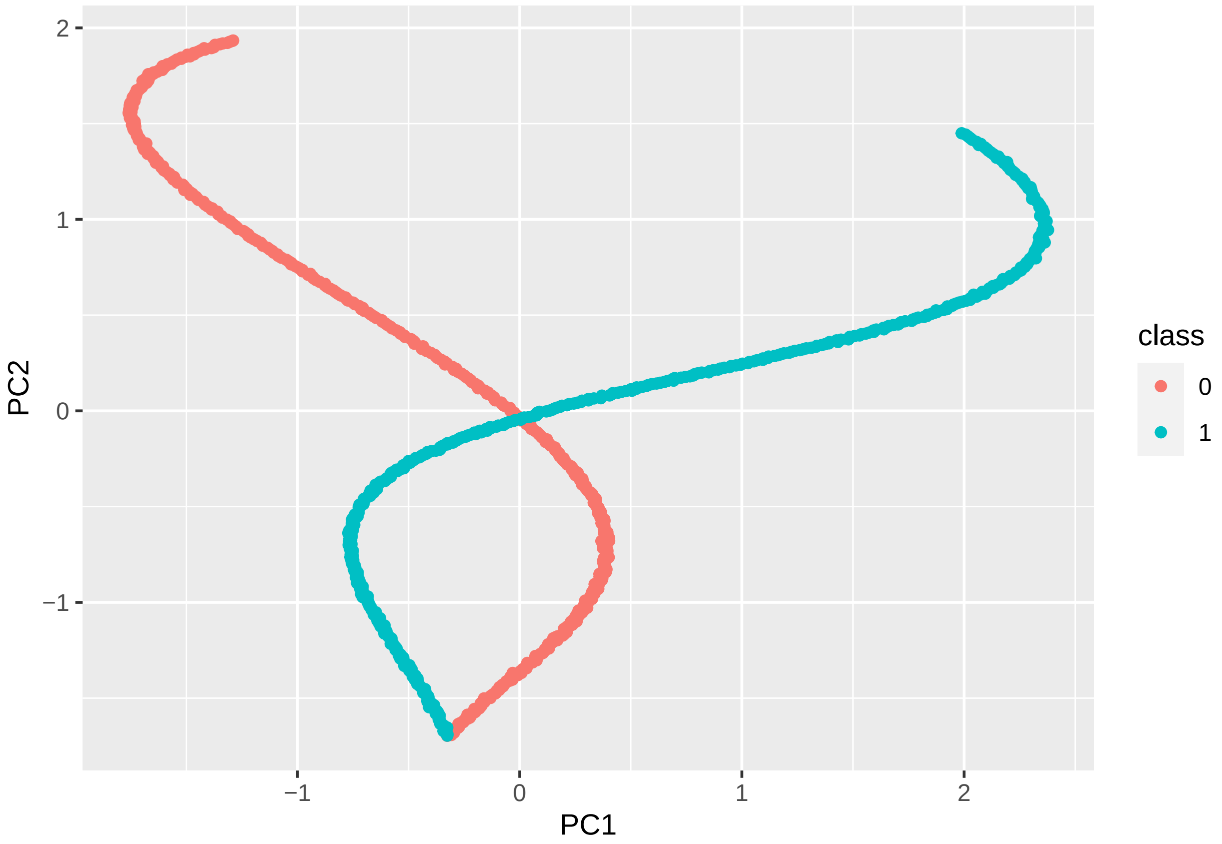
If we perform regular PCA on this dataset, we can either represent the data using just one, two or all three principal components (in the latter case we might as well have not performed PCA).

```
pca = prcomp(Xs, retx=TRUE, rank=1)  # Only use first PC
reduction = data.frame(PC1=pca$x, class=data$class)

ggplot(data = reduction, aes(PC1, factor(0), color = class)) + ylab("") +
  theme(axis.text.y=element_blank(),  #remove y axis labels
        axis.ticks.y=element_blank()  #remove y axis ticks
  ) + geom_point()
```



As we might have expected, a one-dimensional reduction is unable to separate the data. We see slightly better—but far from perfect—separation if we use the first two principal components.

```r
pca = prcomp(Xs, retx=TRUE, rank=2)  # Use first two PCs
reduction = data.frame(PC1=pca$x[,1], PC2=pca$x[,2], class=data$class)

ggplot(data = reduction, aes(PC1, PC2, color = class)) + geom_point()
```



Splitting the dataset into a training and test set, we can see that the classification accuracy of a logistic regression model (trained on the training set) on the test set is only 45.5% when we use just one principal component—worse than flipping a coin. (Note that we perform PCA again but only on the training set—the test set is then reduced using the same principal components found in the training set.)

```r
data = data[sample(1:n, n),]  # Shuffle the data

trainSplit = 0.8
trainIdx = sample(1:n, n*0.8)
train = data[trainIdx,]
test  = data[-trainIdx,]

pca = prcomp(train[,1:3], retx=TRUE)
reduction = data.frame(PC1=pca$x[,1], class=train$class) # Only use first PC
model = glm(class ~ PC1 ,family=binomial(link='logit'), data=reduction)

prediction = predict(model,
                     newdata=data.frame(PC1 = as.matrix(test[,1:3]) %*% pca$rotation[,1]),
                     type="response")
prediction = ifelse(prediction > 0.5,1,0)
mean(prediction == test$class)  # Classification accuracy
```

```
## [1] 0.455
```

Using both principal components results in a classification accuracy of 43.5% (worse than with just 1 PC!), whilst using all three (i.e. using the entire original dataset) achieves 59.5% accuracy. This is a slight improvement, however, we can do much better if we use kernel PCA.

```r
reduction = data.frame(PC1=pca$x[,1], PC2=pca$x[,2], class=train$class) # Use both PCs
model = glm(class ~ PC1 + PC2, family=binomial(link='logit'), data=reduction)

prediction = predict(model,
                     newdata=data.frame(as.matrix(test[,1:3]) %*% pca$rotation),
                     type="response")
prediction = ifelse(prediction > 0.5,1,0)
mean(prediction == test$class)  # Classification accuracy with 2 PCs
```

```
## [1] 0.425
```

```r
reduction = data.frame(PC1=pca$x[,1], PC2=pca$x[,2], PC3=pca$x[,3], class=train$class) # Use all 3 PCs
model = glm(class ~ PC1 + PC2 + PC3, family=binomial(link='logit'), data=reduction)

prediction = predict(model,
                     newdata=data.frame(as.matrix(test[,1:3]) %*% pca$rotation),
                     type="response")
prediction = ifelse(prediction > 0.5,1,0)
mean(prediction == test$class)  # Classification accuracy with all 3 PCs (i.e. original dataset)
```

```
## [1] 0.595
```

## Kernel PCA

To better separate the data we will now use kernel PCA (via the library `kernlab`) with the radial basis kernel $k(x, x') = \exp\left(-\sigma||\mathbf{x} - \mathbf{x}'||^2\right)$ for various bandwidth values $\sigma$. One value of $\sigma$ we will try is the median pairwise distance between all points in the dataset—known as the “median trick”. In our dataset we find that this gives us a value of $\sigma = 2.308164$. We will also investigate the values $\sigma = 0.25$ and $\sigma = 5$ as well as the use of a Laplacian kernel $k(x, x') = \exp\left(-\sigma||\mathbf{x} - \mathbf{x}'||\right)$ where $\sigma$ is chosen with the median trick.

```r
distances = rep(0, n*(n-1)/2)
count = 0
for (i in 1:n-1){
  for (j in i:n){
    count = count + 1
    distances[count] = sqrt(sum((Xs[i, 1:3] - Xs[j,1:3])^2))
  }
}
med = median(distances)
med
```

```
## [1] 2.308164
```

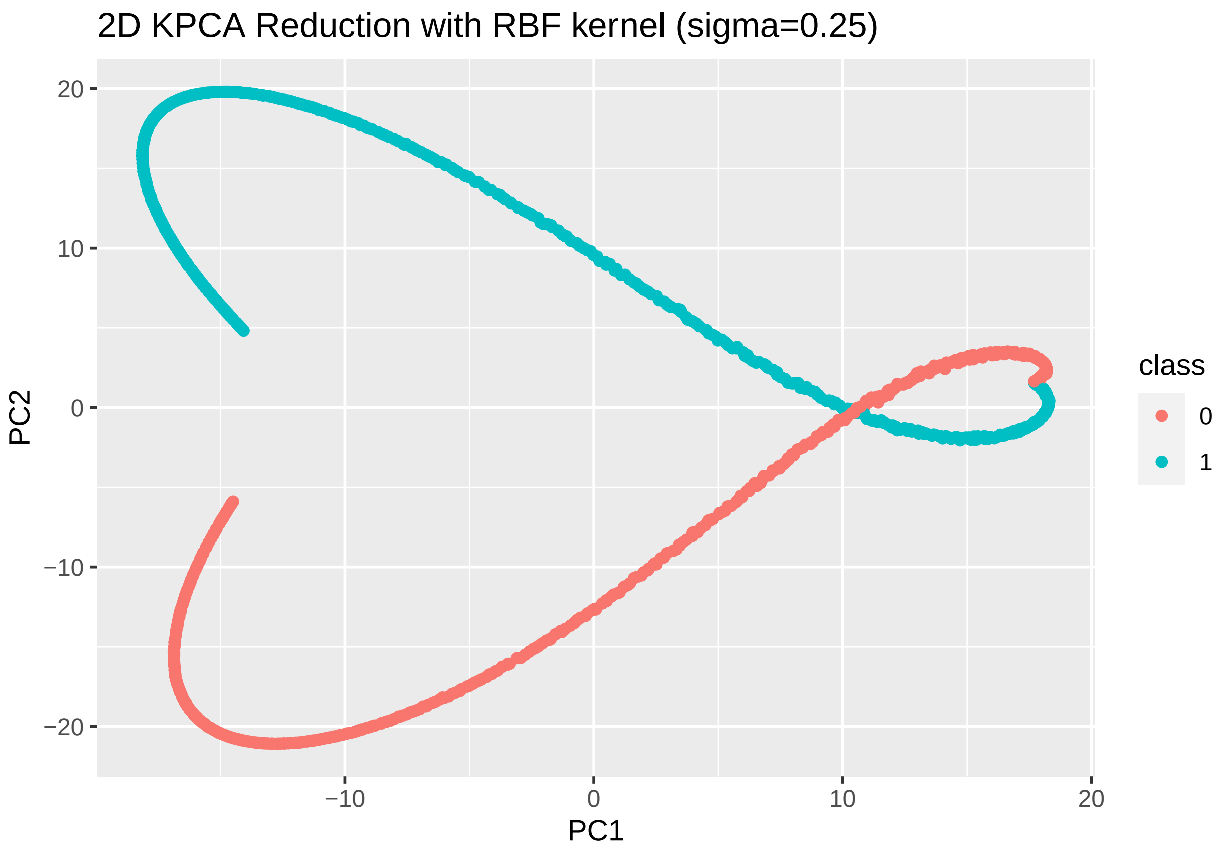
Before we move on to classification with the different kernel bandwidths, we can investigate the 2D PCA reduction of the entire dataset (training and test sets).

```r
library(kernlab)
```

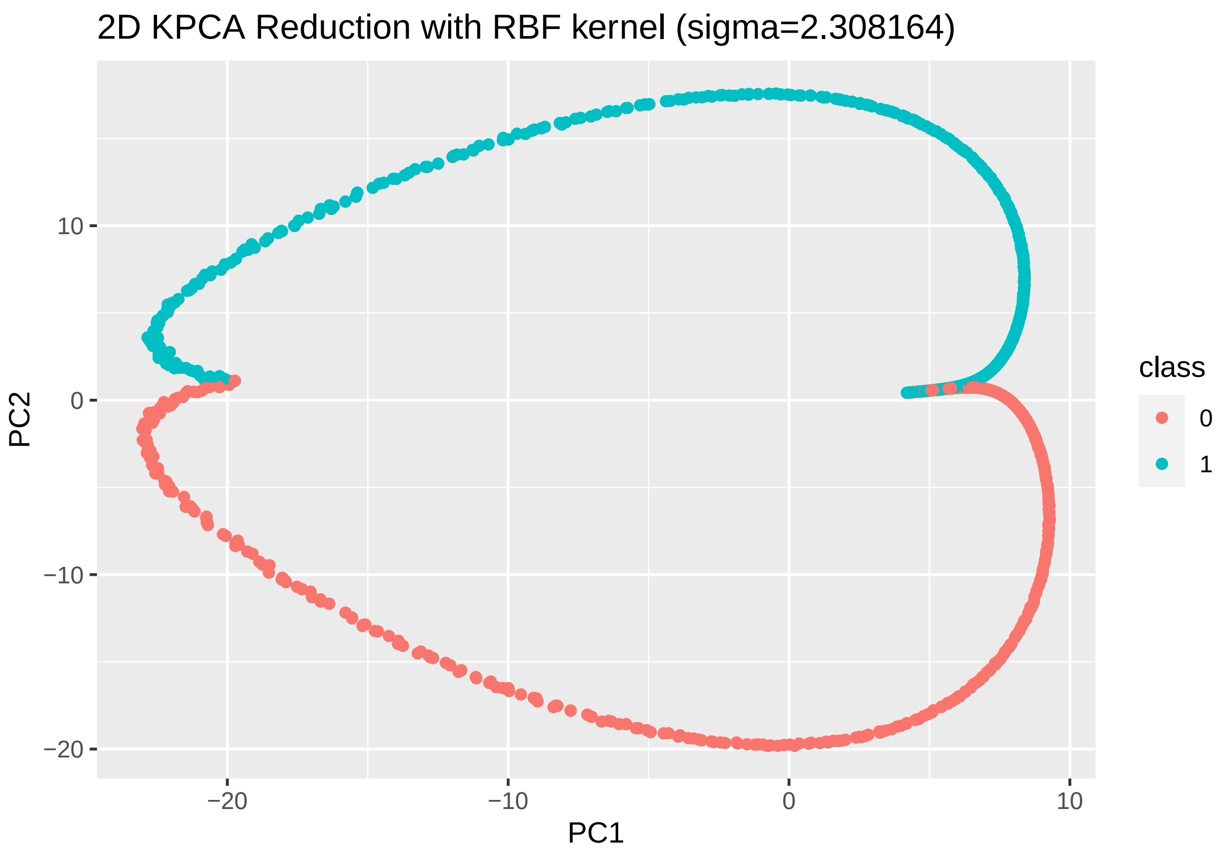
```
##
## Attaching package: 'kernlab'

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

```r
kpca_0.25 = kpca(as.matrix(data[,1:3]), kernel = "rbfdot", kpar = list(sigma = 0.25))
kpca_med = kpca(as.matrix(data[,1:3]), kernel = "rbfdot", kpar = list(sigma = med))
kpca_5 = kpca(as.matrix(data[,1:3]), kernel = "rbfdot", kpar = list(sigma = 5))
```

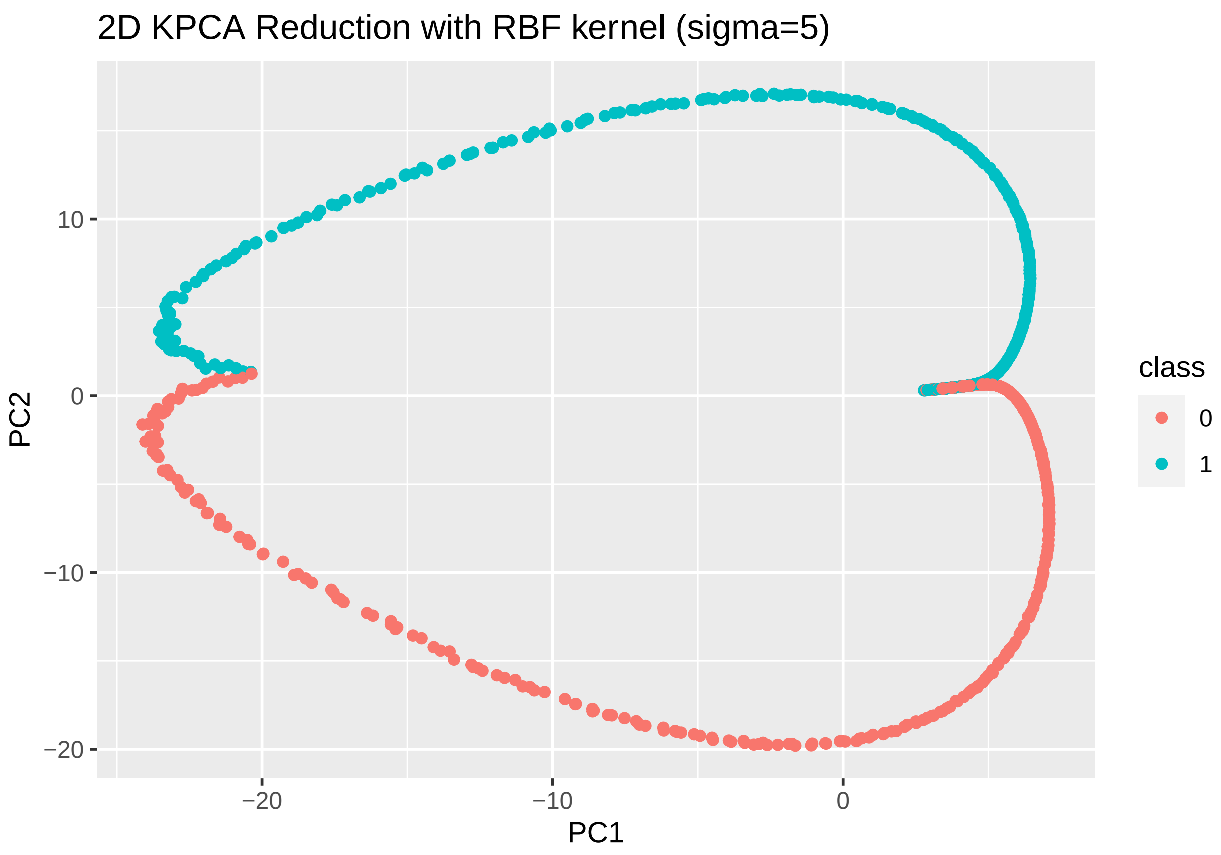
```r
kpca_lapl = kpca(as.matrix(data[,1:3]), kernel = "laplacedot", kpar = list(sigma = med))

reduction = data.frame(PC1=kpca_0.25@rotated[,1], PC2=kpca_0.25@rotated[,2], class=data$class)
ggplot(data = reduction, aes(PC1, PC2, color = class)) +
  ggtitle("2D KPCA Reduction with RBF kernel (sigma=0.25)") + geom_point()
```



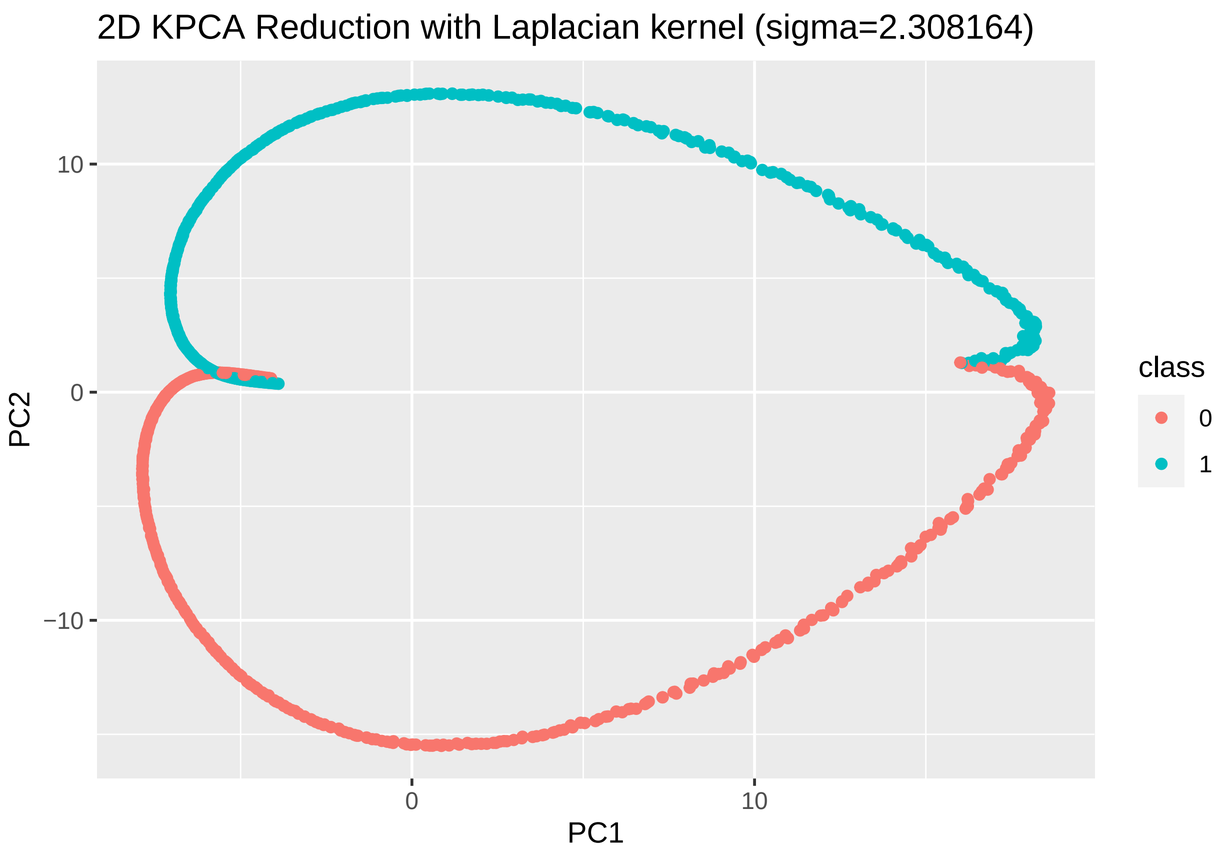
```r
reduction = data.frame(PC1=kpca_med@rotated[,1], PC2=kpca_med@rotated[,2], class=data$class)
ggplot(data = reduction, aes(PC1, PC2, color = class)) +
  ggtitle("2D KPCA Reduction with RBF kernel (sigma=2.308164)") + geom_point()
```



```
reduction = data.frame(PC1=kpca_5@rotated[,1], PC2=kpca_5@rotated[,2], class=data$class)
ggplot(data = reduction, aes(PC1, PC2, color = class)) +
  ggtitle("2D KPCA Reduction with RBF kernel (sigma=5)") + geom_point()
```



```r
reduction = data.frame(PC1=kpca_lapl@rotated[,1], PC2=kpca_lapl@rotated[,2], class=data$class)
ggplot(data = reduction, aes(PC1, PC2, color = class)) +
  ggtitle("2D KPCA Reduction with Laplacian kernel (sigma=2.308164)") + geom_point()
```



These 2D plots seem broadly more linearly separable than the original dataset (and the regular PA reductions thereof), with particularly good-looking (and extremely similar) results for the RBF kernel with $\sigma = 2.308164$ and $\sigma = 5$ as well as for the Laplacian kernel too. Indeed, whilst fitting a simple logistic regression on the first reduction (RBF with $\sigma = 0.25$) gives us a small improvement on our previous-best classification accuracy (61% compared to the 59.5% achieved on the full original dataset), doing the same on the other three reductions improves our results greatly, with an impressive maximum of 91.5% test accuracy when we use the median trick on the Laplacian kernel. (Again, note that we're finding the principle components of the training set and projecting the test set onto these.)

```r
kpca_0.25 = kpca(as.matrix(train[,1:3]), kernel = "rbfdot", kpar = list(sigma = 0.25))
kpca_med = kpca(as.matrix(train[,1:3]), kernel = "rbfdot", kpar = list(sigma = med))
kpca_5 = kpca(as.matrix(train[,1:3]), kernel = "rbfdot", kpar = list(sigma = 5))
kpca_lapl = kpca(as.matrix(train[,1:3]), kernel = "laplacedot", kpar = list(sigma = med))

classificationAccuracy <- function(kpca_obj, numPCs){
  reduction = as.data.frame(kpca_obj@rotated[,1:numPCs])
  reduction$class = train$class
  model = glm(class ~ ., family=binomial(link='logit'), data=reduction)

  testReduction = as.data.frame(predict(kpca_obj, test[,1:3]))[,1:numPCs]
  testReduction$class = test$class

  prediction = predict(model, newdata=testReduction,  type="response")
  prediction = ifelse(prediction > 0.5,1,0)

  mean(prediction == testReduction$class)  # Classification accuracy
}
classificationAccuracy(kpca_0.25, 2)
```

```
## [1] 0.61
```

```
classificationAccuracy(kpca_med, 2)
```

```
## [1] 0.875
```

```
classificationAccuracy(kpca_5, 2)
```

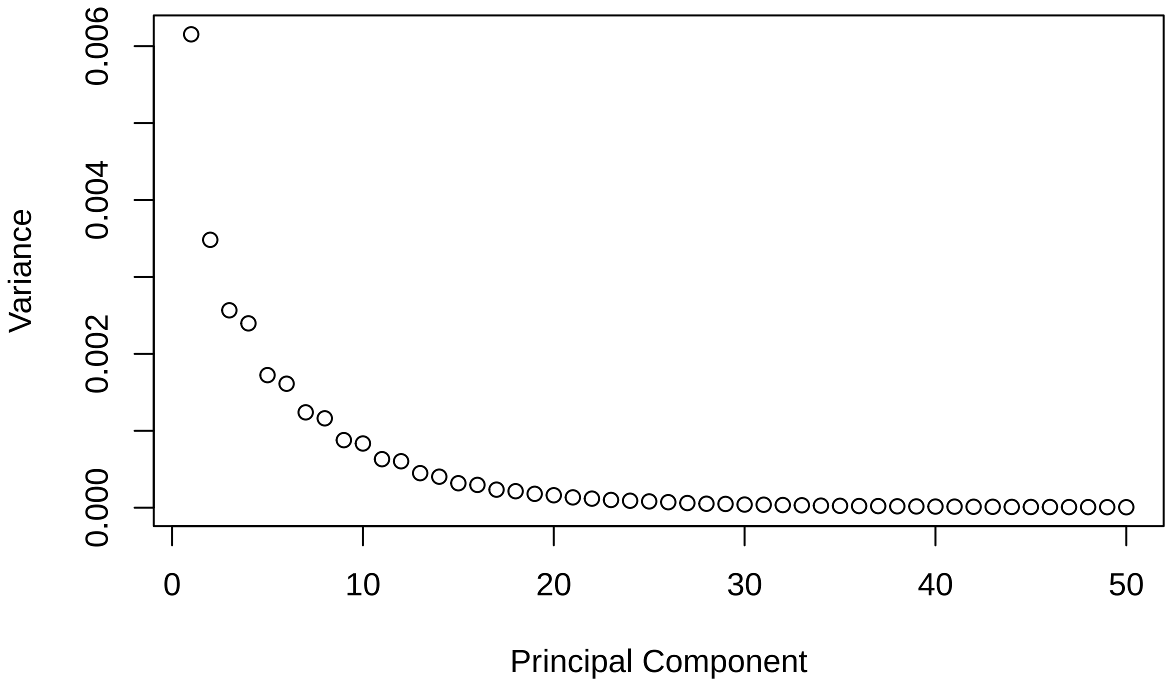
```
## [1] 0.79
```

```
classificationAccuracy(kpca_lapl, 2)
```

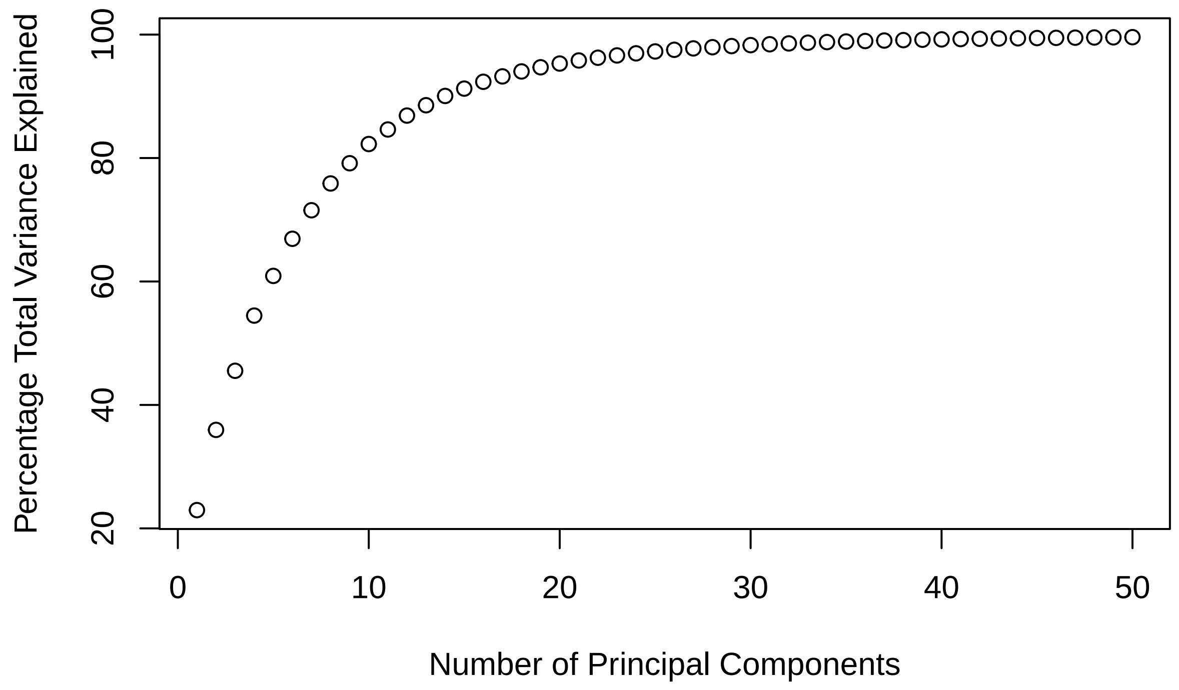
```
## [1] 0.915
```

We'll finish by finding out the classification accuracy when we use more than just the first two principal components with the Laplacian kernel. First we'll make a scree plot of the first 50 principal components in order to decide on how many to keep.

```
maxPCs = 50
plot(1:maxPCs, (kpca_lapl@eig^2)[1:maxPCs], ylab="Variance", xlab="Principal Component")
```



```
totalVar = sum(kpca_lapl@eig^2)
# Or plotting cumulatively explained variance
plot(1:maxPCs, 100*cumsum(kpca_lapl@eig**2 / totalVar)[1:maxPCs],
     ylab="Percentage Total Variance Explained", xlab="Number of Principal Components")
```



Subsequent principal components explain less variance in the dataset than the PCs before them, as expected, however, we can see significant drop-offs in this variance after the 3rd, 5th and 11th PCs. For this reason, we'll finish this investigation by comparing the classification accuracy when using 3, 5, and 11 principle components, which explain 45.5%, 60.9%, and 84.6% of the total variance of the dataset respectively. We'll also try using just 1 principle component (which explains 23.0% of the total variance of the dataset) since this will provide a strong comparison to the one-dimensional classification performed with regular PCA earlier in the portfolio.

```
# % variance explained by first 1/3/5/11 PCs
cumsum(100*kpca_lapl@eig**2 / totalVar)[c(1,3,5,11)]
```

```
##   Comp.1   Comp.3   Comp.5  Comp.11
## 22.96078 45.53084 60.90401 84.62666
```

```
# Classification

# Have to deal with 1 PC case separately
reduction = data.frame(V1 = kpca_lapl@rotated[,1], class = train$class)
model = glm(class ~ ., family=binomial(link='logit'), data=reduction)
testReduction = data.frame(V1 = predict(kpca_lapl, test[,1:3])[,1], class = test$class)
prediction = predict(model, newdata=testReduction,  type="response")
prediction = ifelse(prediction > 0.5,1,0)
mean(prediction == test$class)
```

```
## [1] 0.5
```

```
# 3, 5, 11 cases
classificationAccuracy(kpca_lapl, 3)
```

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

```
classificationAccuracy(kpca_lapl, 5)
```

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

```
classificationAccuracy(kpca_lapl, 11)
```

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

Whilst the one-dimensional case is slightly better than its non-kernel-PCA counterpart (50% compared to 42.5%), it's still only as good as if we predicted the classes uniformly at random. The impressive results come when we look at using more principle components: in all cases of using three or more PCs we achieve an accuracy of 100%.

Although we have been using a relatively small toy dataset, we have shown that kernel PCA can greatly help improve classification accuracy for simple models by allowing us to use principle components in higher dimensions than those of our original dataset. Importantly, we've also seen that (at least on our dataset) we don't need to use all that many of the (potentially-infinite) principal components to obtain a simple model that performs well—recall that using two principal components led to a 91.5% accuracy. Furthermore, it may be possible to further improve upon these results through a more thorough exploration of potential kernels: we could have tried more bandwidth parameters for the RBF and Laplacian kernels and we also could have tried other kernels such as the Bessel or tanh kernels.