Kernel Principal Component Analysis

Sam Berglin [*] Luke Zheng [†] Yuan Ma [‡]
Tianchang Li [§]

May 8, 2019

Abstract

We investigate kernel principal component analysis (kPCA). We introduce the mathematical derivation for kPCA and different kernels commonly used in kPCA. We also include simulation studies to demonstrate the non-linearity of kPCA, the importance of the kernel function, and significance of kernel hyperparameters. We also conduct kPCA on a simulated higher dimensional dataset.

[*]Department of Statistics. Email: berglin@wisc.edu
[†]Department of Statistics. Email: lzheng54@wisc.edu
[‡]Department of Statistics. Email: ma227@wisc.edu
[§]Department of Statistics. Email: tli289@wisc.edu

1 Introduction: Principal Component Analysis

A higher dimensional dataset can be difficult to interpret. PCA effectively reduces the dimension so that we can focus on the most important features. The first "principal component" is a normalized linear combination of the features that has the largest variance in the dataset. Each successive principal component has the next highest variance, so earlier principal components explain larger amounts of the variance. These early linear principal components can form a new, lower dimensional perspective of the data while retaining a maximal amount of information, in the form of variance.

The principal components are found by solving for the data's covariance matrix's eigenvalues and eigenvectors. The eigenvectors that correspond to larger eigenvalues correspond with the principal components that explain the most variance. So the eigenvector with the largest eigenvalue is the first principal component which explains the most variance among all of the components.

Common notation of normal PCA:

$$Z_i = \phi_{i1} X_1 + ... + \phi_{id} X_d \tag{1}$$

where each loading vector (ϕ_i) is orthogonal to the previous ones.

In practice, assume we have a high dimensional data matrix $X \in \mathbb{R}^{n \times d}$. We wish to project this dataset onto a lower dimensional space while preserving as much of the variability in the data as possible by linearly separating the data on that dimension. We define each dimension of this lower dimensional space to be a linear combination of the original space.

2 Extension to Kernel Principal Component Analysis

Kernel principle component analysis (kPCA) is an extension of principle component analysis (PCA) using kernel methods to perform the linear operations of PCA in a reproducing kernel Hilbert space. Since PCA cannot explain non-linear structures of datasets, kernel PCA is developed to find non-linear, low dimensional manifolds. In fact, PCA can be considered a special case of kPCA where the kernel space is just the linear kernel.

The idea of kPCA is simple. Although m points cannot in general be linearly separated in $d \leq m$ dimensions, they can almost always be linearly

separated in higher dimensions. Therefore, instead of using the points in the original space as x_i, we want to go to some higher-dimensional feature space as $\phi(x_i) \in \mathbb{R}^N$. Then we can do PCA on the higher dimensional space and return it to a lower dimensional space. This allows kPCA to non-linearly separate data in the original, lower dimensional space.

3 Mathematical Formulation

We formulate kPCA in a mathematical context. We base our formulation from work by Precup (2018). For simplicity, we center the data in the feature space so the mean is zero, so $\sum_{i=1}^{m} \phi(x_i) = 0$. This can always be done on a real dataset.

Assume we have the covariance matrix

$$C = \frac{1}{m} \sum_{i=1}^{m} \phi(x_i)\phi(x_i)^T. \tag{2}$$

The eigenvectors $v_1, ..., v_j$ and eigenvalue $\lambda_1, ..., \lambda_j$ satisfy

$$Cv_j = \lambda_j v_j, j = 1, ...N. \tag{3}$$

To avoid explicitly going to feature space, we use kernels.

$$K(x_i, x_k) = \phi(x_i)^T \phi(x_k) \tag{4}$$

Re-writing the PCA equation, we get

$$\frac{1}{m} \sum_{i=1}^{m} \phi(x_i)\phi(x_i)^T v_j = \lambda_j v_j, j = 1, ...N. \tag{5}$$

The eigenvectors can also be written as linear combinations of features.

$$v_j = \sum_{i=1}^{m} a_{ji}\phi(x_i) \tag{6}$$

Finding the eigenvectors is equivalent to finding the coefficients a_{ij}.

By substituting Equation 5 back into the equation we get

$$\frac{1}{m} \sum_{i=1}^{m} \phi(x_i)\phi(x_i)^T (\sum_{l=1}^{m} a_{jl}\phi(x_l)) = \frac{1}{m} \sum_{i=1}^{m} \phi(x_i)(\sum_{l=1}^{m} a_{jl}K(x_i, x_l)) = \lambda_j \sum_{l=1}^{m} a_{jl}\phi(x_l) \tag{7}$$

By multiplying Equation 6 by $\phi(x_k)^T$ to the left and plugging in the kernel again, we get

$$\frac{1}{m}\sum_{i=1}^{m} K(x_k, x_i)(\sum_{l=1}^{m} a_{jl} K(x_i, x_l)) = \lambda_j \sum_{l=1}^{m} a_{jl} K(x_k, x_l), \forall j, k. \tag{8}$$

Rearranging, we get $K^2 a_j = m\lambda_j K a_j$. We can remove a factor of K from both sides to get

$$K a_j = m\lambda_j a_j. \tag{9}$$

Note that removing K will only affect eigenvectors with eigenvalues 0, which are not considered as principal components.

We have a normalization condition for the a_j vectors:

$$v_j^T v_j = 1 \Rightarrow \sum_{k=1}^{m}\sum_{l=1}^{m} a_{jl} a_{jk} \phi(x_l)^T \phi(x_k) = 1 \Rightarrow a_j^T K a_j = 1. \tag{10}$$

Plugging Equation 9 into Equation 8, we get

$$\lambda_j m a_j^T a_j = 1, \forall j. \tag{11}$$

For a new point x, its projection onto the principal components is

$$\phi(x)^T v_j = \sum_{i=1}^{m} a_{ji} \phi(x)^T \phi(x_i) = \sum_{i=1}^{m} a_{ji} K(x, x_i). \tag{12}$$

In general, the features $\phi(x_i)$ may not have mean 0, so we need to normalize the feature space (centralize K). We will work with

$$\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{m}\sum_{k=1}^{m} \phi(x_k). \tag{13}$$

The corresponding kernel matrix entries are given by

$$\tilde{K}(x_k, x_l) = \tilde{\phi}(x_l)^T \tilde{\phi}(x_j). \tag{14}$$

By doing some algebra, we get

$$\tilde{K} = K - 2I_{\frac{1}{m}} K + I_{\frac{1}{m}} K I_{\frac{1}{m}}, \tag{15}$$

where $I_{\frac{1}{m}}$ is the matrix with all elements equal to $\frac{1}{m}$.

3.1 Summary

To summarize, there are four steps for the Kernel PCA method.

1. Pick a kernel function.
2. Construct the normalized kernel matrix $\tilde{K}$ for the data.
3. Find the eigenvalues λ_j and eigenvectors v_j for the matrix.
4. Represent a data point as the following set of features:

$$y_j = \sum_{i=1}^{m} a_{ji} K(x, x_i), j = 1, ...m. \tag{16}$$

3.2 Common Kernel Functions

Some common kernel functions include

- Gaussian: $K(x, x') = \exp(\frac{-||x-x'||}{2\sigma^2})$
- Laplacian: $K(x, x') = \exp\left(\frac{-||x-x'||}{\sigma}\right)$
- Spline (x_i is the i^{th} entry of $x \in \mathbb{R}^d$):

$$K(x, x') = \prod_{i=1}^{d} 1 + x_i x_i' + x_i x_i' \min(x_i, x_i') - \frac{x_i + x_i'}{2} \min(x_i, x_i')^2 + \frac{\min(x_i, x_i')^3}{3}$$

A kernel function is meant to represent similarity between two points. The higher the kernel is, the more similar or "close" the points are in the Reproducing Kernel Hilbert Space.

4 Simulations

4.1 Non-linear Separation with "Moons" Data

We show that we can make non-linearly separable data linearly separable through kPCA on the "Moons" dataset. The data is two dimensional with two classes (red and black) and illustrated in Figure 1. We simulate it in our R code. Inspiration for this simulated data came from scikit learn (2018b).

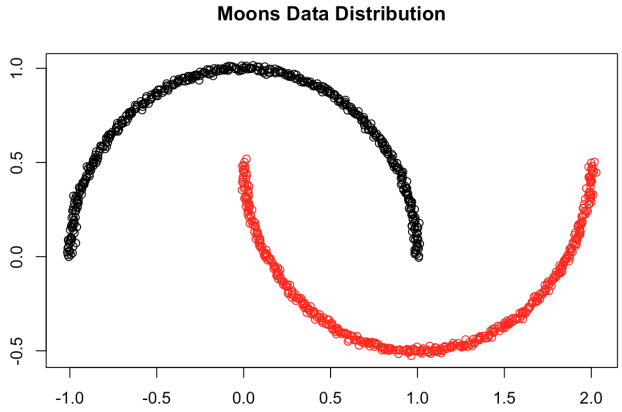
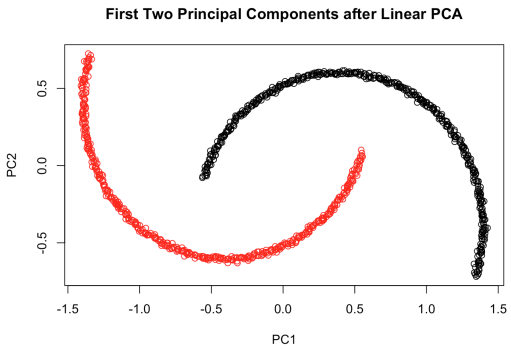


Figure 1: Moons data.

We apply PCA in Figure 2. Each axis is one of the first two principal components. We see the data still remains inseparable by a line.




```
## Importance of components:
##                             PC1    PC2
## Standard deviation     0.9041 0.4233
## Proportion of Variance 0.8202 0.1798
## Cumulative Proportion  0.8202 1.0000
```

Figure 2: Linear PCA on Moons data. The component importance statistics show the proportion of total variance carried by each principal component.

Now, we apply kPCA to see if it can linearly separate the two classes in the kernel space. We apply the Gaussian kernel with parameter σ defined as

$$K(x, x') = \exp\left(\frac{-||x - x'||^2}{2\sigma^2}\right) \tag{17}$$

We set $\sigma = 15$ and postpone discussion of kernel hyperparameters to the next section for simplicity. The results are shown in Figure 3. Once

again, the first two principal components (in the kernel space) are shown in the left plot. All kPCA plots with these colors have the first and second principal components on the x and y axes, respectively. Clearly, they are now linearly separable in the kernel space by a vertical line. This is equivalent to a non-linear separation in the original feature space. The right plot shows the proportion of variance carried by each principal component. Notice how we have over 50 principal components despite our two dimensional dataset. This is due to the "kernel" trick.

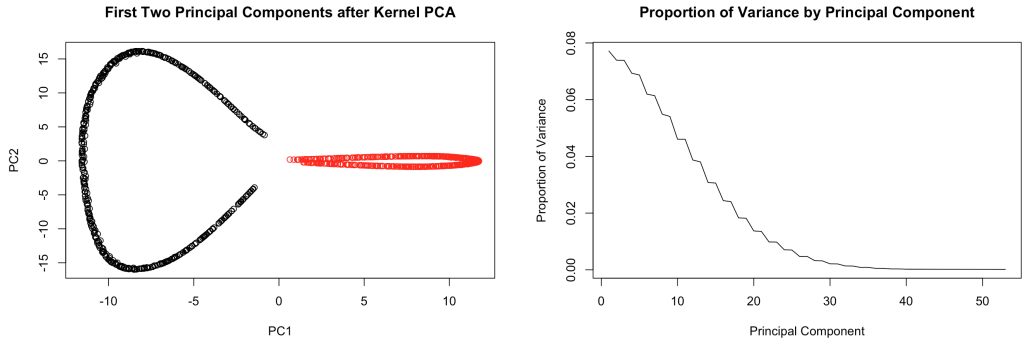


Figure 3: kPCA on Moons. The left plot shows the data distribution on the first two components of the kPCA results. The right plot shows the proportion of variance carried by each principal component.

4.2 Varying Hyperparameters

Thus far, the kernel we have used is the Gaussian kernel. It represents "similarity" between points based on their distance from one another. We demonstrate the importance of the parameter σ in this kernel on our simulated "Circles" dataset in Figure 4. Inspiration for this dataset's simulation came from scikit learn (2018a).

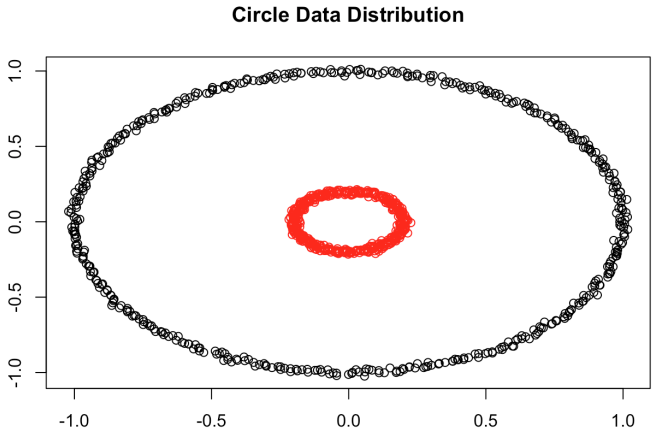


Figure 4: "Circles" dataset.

Now we vary σ across three values ($0.1, 1$, and 20) to demonstrate the effect of σ. We show all three plots in Figure 5.

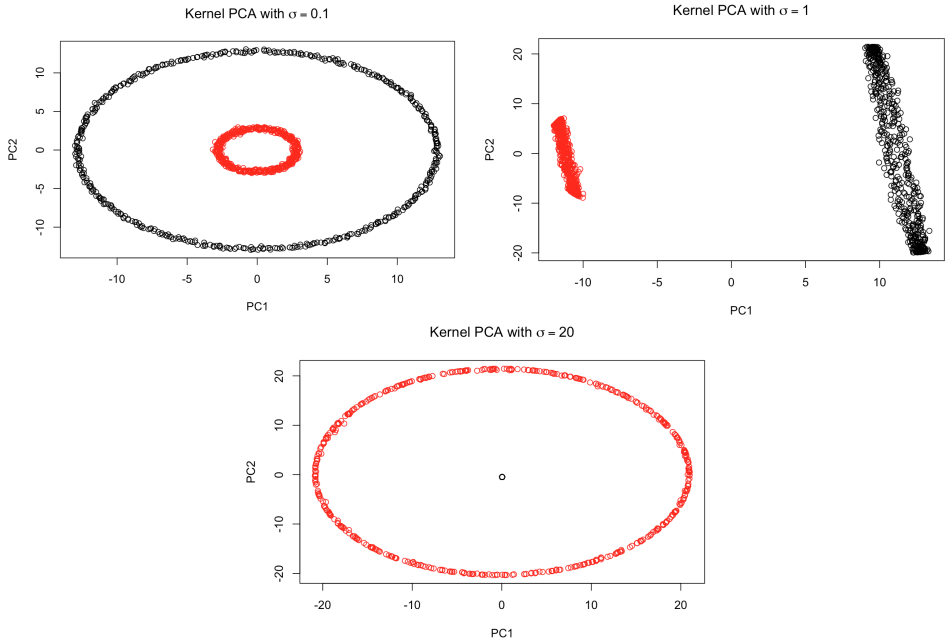


Figure 5: kPCA with $\sigma = 0.1, 1, 20$. Once again, all colored kPCA plots have the first and second principal components on the x and y axes, respectively.

We see that $\sigma = 0.1$ and $\sigma = 20$ fail to linearly separate the data, but the "sweet spot" of $\sigma = 1$ works well. σ is considered a hyperparameter of the kernel function and must be learned with a performance metric based on the application. For example, if we are using kPCA to project the data and are then building a classifier from the first five principal components, we would use a cross-validated accuracy estimate to select the optimal value of σ. For the purpose of these plots, we visually assessed linear separability to select the three values of σ we displayed.

4.3 New Kernels

Various kernel function besides the Gaussian kernel can also be used. Choice of such a kernel can also be considered a hyperparameter. Selection of the kernel function can also be selected with cross-validation, as discussed for σ in the Gaussian kernel.

For our examination of various kernel functions, we stick with the Circles

dataset. We begin with the simplest kernel function: the linear kernel.

$$K(x, x') = x^T x' \quad x, x' \in \mathbb{R}^d \tag{18}$$

We apply it and the results in Figure 6 suggests that the linear kernel does not linearly separate the data.

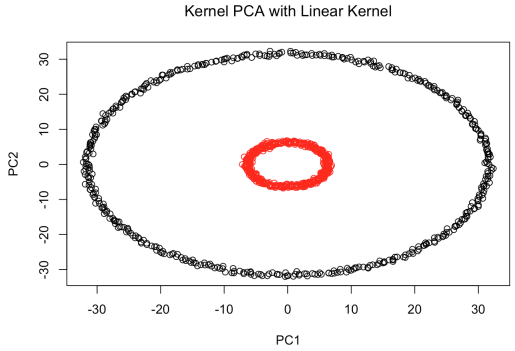


Figure 6: kPCA on the circles dataset with linear kernel.

However, this is expected! kPCA with a linear kernel is, in fact, just standard PCA. If we apply PCA, we get the following plot in Figure 7. These results are identical and expected.

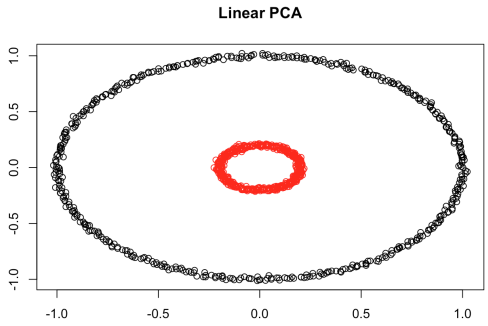


Figure 7: PCA on circles dataset.

Now, we consider the Laplacian kernel. It is similar to the Gaussian

kernel and is defined as

$$K(x, x') = \exp\left(\frac{-||x - x'||}{\sigma}\right) \tag{19}$$

Since the Gaussian kernel provided good separation, we expect the Laplacian kernel to have similar results. They are displayed in Figure 8. As expected, the distribution is similar to that of the Gaussian kernel and the Laplacian kernel provides strong linear separation.

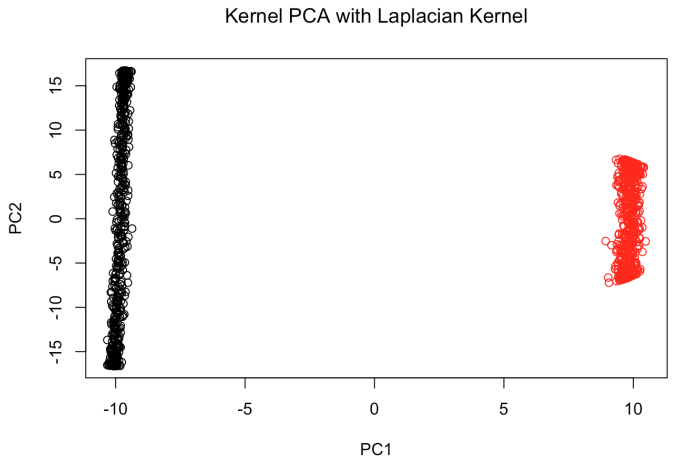


Figure 8: Laplacian kPCA.

The final kernel we apply in our simulations is the spline kernel. If we let x_i be the i^{th} entry of $x \in \mathbb{R}^d$, the spline kernel is defined by

$$K(x, x') = \prod_{i=1}^{d} 1 + x_i x_i' + x_i x_i' \min(x_i, x_i') - \frac{x_i + x_i'}{2} \min(x_i, x_i')^2 + \frac{\min(x_i, x_i')^3}{3} \tag{20}$$

and the results are given in Figure 9. Note that the spline kernel has no hyperparameters. The spline kernel distorts the data from its original concentric circles, but does not linearly separate them.

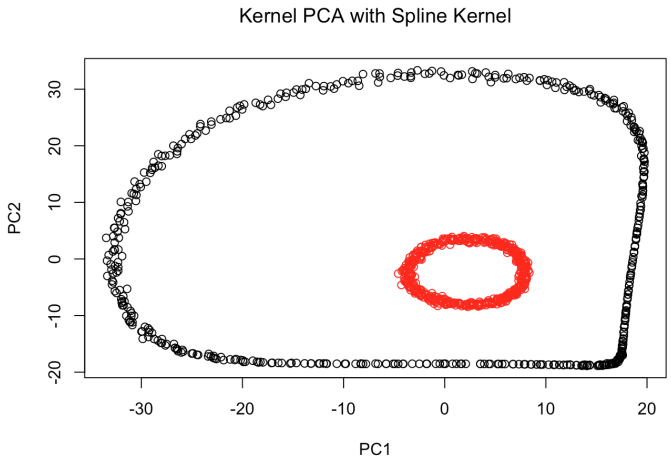


Figure 9: Spline kPCA.

Our simulations have suggested that the Gaussian and Laplacian kernels were the most effective on the Circles dataset. This is reasonable since they are the two widely used kernel functions according to James et al. (2013).

4.4 Discovering Manifolds in Higher Dimensions

In our previous simulations, we dealt with only two dimensional datasets. Now, we simulate higher dimensional data to see if kPCA can still find separations in the high-dimensional space. To do this, we simulate two dimensional data that we know can be linearly separated (in the high-dimensional space) with kPCA and add three "nuisance" variables. These are non-linear functions of the first two variables with added noise. We then apply kPCA on this five dimensional dataset. We expect that kPCA will still be able to separate the two classes despite the nuisance dimensions.

Specifically, we draw X_1 and X_2 from the simulated Moons dataset. Then, we define the next three variables as follows:

$$
\begin{aligned}
X_3 &= \text{sign}(X_1) + 3X_2 + \varepsilon \\
X_4 &= -\exp(X_1) + X_2^2 + \varepsilon \\
X_5 &= \sin(X_1) + \cos(X_2) + \varepsilon \\
\varepsilon &\sim \text{N}(\mu = 0, \sigma = 0.25)
\end{aligned}
$$

As a baseline, we apply PCA on this dataset and the results are displayed in Figure 10. We use squares instead of circles so we can more accurately assess separation. However, this is not an issue in Figure 10 since it fails to even recover the true sharp "moon" structure in the original data.

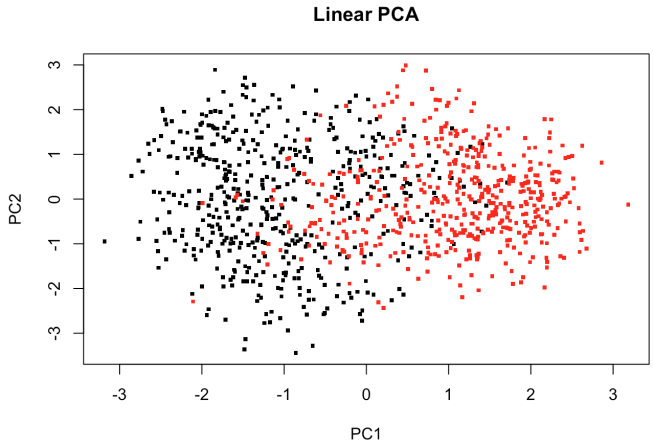


Figure 10: PCA on high-dimensional data.

Now, we apply kPCA with a Gaussian kernel where $\sigma = 3$. Note that σ was selected through visually assessing the data distribution in the two-dimensional kernel space. The results are in Figure 11. We use slightly transparent points to more accurately visualize point density.

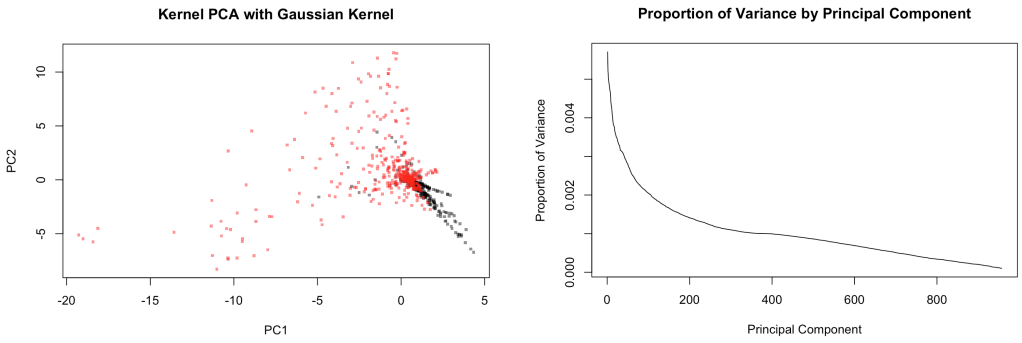


Figure 11: kPCA on high dimensional data with Gaussian kernel ($\sigma = 3$).

Here the Gaussian kernel is still able to somewhat linearly separate the

data, despite the noise. The red class remains in the top left of the plot while the black class is concentrated in the lower right. However, the separation is much less clear than for the original moons data. The plot of the proportion of variance still shows a decreasing trend similar to the two-dimensional data in Figure 3.

We also apply kPCA with Laplacian kernel ($\sigma = 3$) and a spline kernel in Figures 12 and 13, respectively.

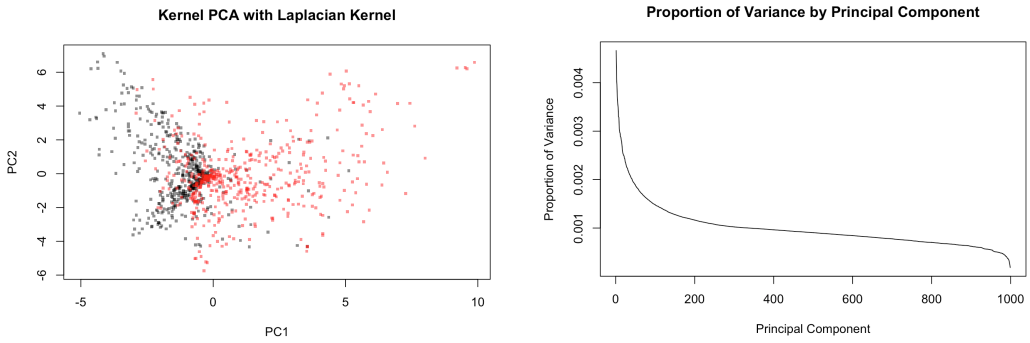


Figure 12: kPCA on high dimensional data with Laplacian kernel ($\sigma = 3$).

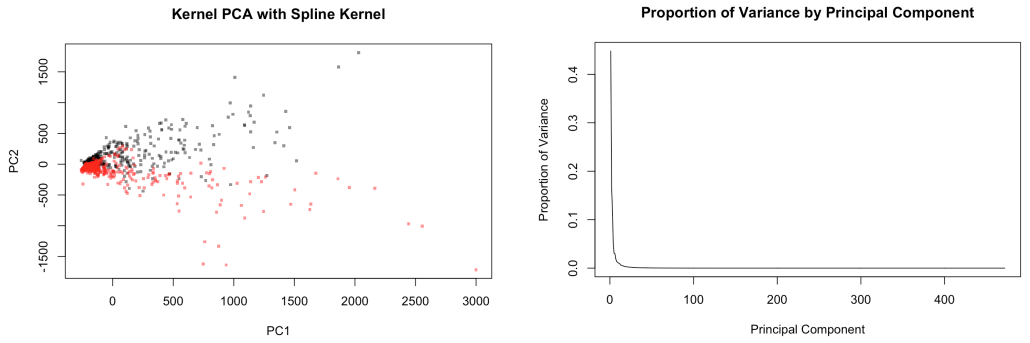


Figure 13: kPCA on high dimensional data with spline kernel.

The Laplacian kernel, once again similar to Gaussian, can still somewhat separate the data. However, it is not a clear separation by any means. The spline kernel still linearly separates better than PCA, but the separation is inferior to the Laplacian and Gaussian kernels.

5 Conclusions

Since kPCA can do non-linear dimension reduction, non-linear data can be separated with the correct kernel. However, to do so requires selecting an appropriate kernel and hyperparameters. This requires a test dataset or prior knowledge of the data. However, supposing one has enough knowledge of the data or of an appropriate kernel, kPCA can be a powerful classification tool due to its ability to separate non-linear data.

Nevertheless, there are some weaknesses of kPCA. Unlike regular PCA, kPCA's output is more difficult to interpret since we have far more principal components. As shown through the mathematical derivation, kPCA maps the data into a higher dimensional space, and then conducts PCA in this unintuitive space. Hence, the output of kPCA is trickier to use for inference. In regular PCA, the loadings for each principal component have very clear and useful meaning; they are linear combinations of the original features. However in kPCA, loadings have less meaning since they are from a higher dimensional space. Computationally, kPCA can be expensive for larger datasets since it relies matrices of size $m \times m$, where m is the number of observations in the dataset. This can be alleviated with subsampling.

The technique of kPCA can be used in many modern applications due to its potential for solving non-linear classification problems. For example, Mika et al. (1999) use kPCA to de-noise data. Furthermore, Kim et al. (2002) use kPCA for facial recognition.

5.1 Group Member Contributions

Tianchang Li and Yuan Ma collaborated to put together the mathematical derivations and motivations behind kPCA. Luke Zheng and Sam Berglin collaborated on the R code for all simulations.

References

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). An introduction to statistical learning, volume 112. Springer.

Kim, K. I., Jung, K., and Kim, H. J. (2002). Face recognition using kernel principal component analysis. IEEE signal processing letters, 9(2):40–42.

Mika, S., Schölkopf, B., Smola, A. J., Müller, K.-R., Scholz, M., and Rätsch, G. (1999). Kernel pca and de-noising in feature spaces. In Advances in neural information processing systems, pages 536–542.

Precup, D. (2018). Dimensionality reduction. pca. kernel pca. https://www.cs.mcgill.ca/ dprecup/courses/ML/Lectures/ml-lecture13.pdf.

scikit learn (2018a). Circles dataset. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make circles.html.

scikit learn (2018b). Moons dataset. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make moons.html.

Appendix: R code

```
library(kernlab)
library(latex2exp)
library(knitr)
library(scales)
set.seed(2019)

# Generating Moons Data

Generation code largely based off of ‘make_moons‘ from ‘scikit-learn‘

make_moons = function(n = 1000, noise = 0.01) {

    n_out = n %/% 2
    n_in = n - n_out

    outer_circ_x = cos(seq(0, pi, length.out = n_out)) +
        rnorm(n = n_out, sd = noise)
    outer_circ_y = sin(seq(0, pi, length.out = n_out)) +
        rnorm(n = n_out, sd = noise)
    inner_circ_x = 1 - cos(seq(0, pi, length.out = n_in)) +
        rnorm(n = n_in, sd = noise)
    inner_circ_y = 1 - sin(seq(0, pi, length.out = n_in)) - .5 +
        rnorm(n = n_in, sd = noise)

    X = t(rbind(c(outer_circ_x, inner_circ_x), c(outer_circ_y, inner_circ_y)))
    Y = c(rep(1, n_out), rep(2, n_in))

    return(list(X = X, Y = Y))
}

moons = make_moons()
plot(moons$X, col = moons$Y, xlab = "", ylab = "",
     main = "Moons Data Distribution")

# Linear PCA

PCA = prcomp(moons$X, retx = TRUE)
summary(PCA)
plot(PCA$x, col = moons$Y,
     main = "First Two Principal Components after Linear PCA")
plot(cbind(PCA$x[,1], jitter(rep(0, nrow(PCA$x)))), xlab = "PC1", col = moons$Y,
     ylab = "",
     main = "First Principal Component after Linear PCA", ylim = c(-0.2, 0.2))
```

```
# Kernel PCA

var.prop.plot.kpca = function(kpca) {
    x = eig(kpca) / sum(eig(kpca))
    plot(1:length(x), x, xlab = "Principal Component", type = "l",
         ylab = "Proportion of Variance",
         main = "Proportion of Variance by Principal Component")
}

summary.kpca = function(kpca) {
    eigs = kpca@eig
    cum.prop.var = cumsum(eigs) / sum(eigs)
    prop.var = eigs / sum(eigs)
    summary = data.frame(matrix(c(prop.var, cum.prop.var),
                                ncol = length(prop.var), byrow = TRUE))
    colnames(summary) = names(eigs)
    rownames(summary) = c("Proportion of Variance", "Cumulative Proportion")
    return(summary)
}

kPCA = kpca(moons$X, kpar = list(sigma = 15))
plot(kPCA@rotated[,1:2], col = moons$Y, xlab = "PC1", ylab = "PC2",
     main = "First Two Principal Components after Kernel PCA")
kable(summary.kpca(kPCA))
plot(cbind(kPCA@rotated[,1], jitter(rep(0, nrow(PCA$x)))),
     xlab = "PC1", col = moons$Y, ylab = "",
     main = "First Principal Component after Kernel PCA", ylim = c(-0.2, 0.2))
var.prop.plot.kpca(kPCA)

# Generating Circle Data

make_circles = function(n = 1000, factor = 0.2, noise = 0.01) {

    if (factor >= 1 | factor < 0) {
        warning("Factor has to be between 0 and 1.")
    }

    n_out = n %/% 2
    n_in = n - n_out
    noise = 0.01

    linspace_out = seq(0, 2*pi, length.out = n_out)
    linspace_in = seq(0, 2*pi, length.out = n_in)
    outer_circ_x = cos(linspace_out) + rnorm(n = n_out, sd = noise)
```

```
    outer_circ_y = sin(linspace_out) + rnorm(n = n_out, sd = noise)
    inner_circ_x = cos(linspace_in) * factor + rnorm(n = n_in, sd = noise)
    inner_circ_y = sin(linspace_in) * factor + rnorm(n = n_in, sd = noise)

    X = t(rbind(c(outer_circ_x, inner_circ_x), c(outer_circ_y, inner_circ_y)))
    Y = c(rep(1, n_out), rep(2, n_in))
    return(list(X = X, Y = Y))
}

circles = make_circles()
plot(circles$X, col = circles$Y, xlab = "", ylab = "",
     main = "Circle Data Distribution")

# Testing Values of sigma

# Sigma = 0.1
kPCA = kpca(circles$X, kpar = list(sigma = .1))
plot(kPCA@rotated[,1:2], col = circles$Y, xlab = "PC1", ylab = "PC2",
     main = TeX("Kernel PCA with $\\sigma = 0.1$"))

# Sigma = 1
kPCA = kpca(circles$X, kpar = list(sigma = 1))
plot(kPCA@rotated[,1:2], col = circles$Y, xlab = "PC1", ylab = "PC2",
     main = TeX("Kernel PCA with $\\sigma = 1$"))

# Sigma = 20
kPCA = kpca(circles$X, kpar = list(sigma = 20))
plot(kPCA@rotated[,1:2], col = circles$Y, xlab = "PC1", ylab = "PC2",
     main = TeX("Kernel PCA with $\\sigma = 20$"))

# Different Kernels

# Linear Kernel
kPCA = kpca(circles$X, kernel = "vanilladot", kpar = list())
plot(kPCA@rotated[,1:2], col = circles$Y, xlab = "PC1", ylab = "PC2",
     main = TeX("Kernel PCA with Linear Kernel"))

# Exact same as Linear PCA
PCA = prcomp(circles$X, retx = TRUE)
plot(PCA$x, col = circles$Y, main = "Linear PCA", xlab = "", ylab = "")

# Laplacian Kernel
kPCA = kpca(circles$X, kernel = "laplacedot", kpar = list(sigma = 1))
plot(kPCA@rotated[,1:2], col = circles$Y, xlab = "PC1", ylab = "PC2",
     main = TeX("Kernel PCA with Laplacian Kernel"))
```

```
# Spline Kernel
kPCA = kpca(circles$X, kernel="splinedot", kpar=list())
plot(kPCA@rotated[,1:2], col = circles$Y, xlab = "PC1", ylab = "PC2",
     main = TeX("Kernel PCA with Spline Kernel"))

# Discovering Manifolds on Higher Dimensional Data

# Simulating data from circles data
hd = data.frame(moons)
n = nrow(hd)
hd$X.3 = 3*hd$X.2 + sign(hd$X.1) + rnorm(n, sd = 1)
hd$X.4 = -exp(hd$X.1) * hd$X.2^2 + rnorm(n, sd = 1)
hd$X.5 = sin(hd$X.1) + cos(hd$X.2) + rnorm(n, sd = 1)

# Setup for PCA
hd$Y = NULL

# Regular pca
PCA = princomp(as.matrix(hd), cor = TRUE)
summary(PCA)
plot(PCA$scores, col = circles$Y, xlab = "PC1", ylab = "PC2",
     main = "Linear PCA", pch = ".", cex = 4)

# kpca with gaussian kernel
kPCA = kpca(as.matrix(hd), kpar = list(sigma = 3))
kable(summary.kpca(kPCA))
plot(kPCA@rotated[,1:2], xlab = "PC1", ylab = "PC2",
     main = "Kernel PCA with Gaussian Kernel", pch = ".", cex = 4,
     col = alpha(circles$Y, 0.5))
var.prop.plot.kpca(kPCA)

# kpca with laplacian kernel
kPCA = kpca(as.matrix(hd), kpar = list(sigma = 3), kernel = "laplacedot")
kable(summary.kpca(kPCA))
plot(kPCA@rotated[,1:2], xlab = "PC1", ylab = "PC2",
     main = "Kernel PCA with Laplacian Kernel", pch = ".", cex = 4,
     col = alpha(circles$Y, 0.5))
var.prop.plot.kpca(kPCA)

# kpca with spline kernel
kPCA = kpca(as.matrix(hd), kpar = list(), kernel = "splinedot")
kable(summary.kpca(kPCA))
plot(kPCA@rotated[,1:2], xlab = "PC1", ylab = "PC2",
     main = "Kernel PCA with Spline Kernel", pch = ".", cex = 4,
```

```
     col = alpha(circles$Y, 0.5))
var.prop.plot.kpca(kPCA)
```