PCA and Kernel PCA for Wine Variety Analysis

Huanchao Qin
Department of Statistics

University of Auckland

Abstract

Principal component analysis (PCA) is a powerful technique for linear dimensionality reduction and feature extraction. Kernel PCA is a nonlinear version of PCA that effectively takes advantage of the complex spatial structure of high-dimensional features. This report investigates their application in analyzing a wine dataset and determining the most suitable approach for the dataset. It consists of chemical analysis results from three different cultivars, comprising 178 observations with 14 variables. The study compares the performance of standard PCA and various kernel PCA methods, considering classification accuracy and the number of principal components. The results demonstrate that all methods achieved accuracy rates exceeding 95%, with the Sigmoid kernel PCA exhibiting the highest accuracy (100%). However, it required a large number of principal components. Conversely, the Laplacian kernel PCA achieved comparable accuracy with significantly fewer principal components. The findings highlight the effectiveness of kernel PCA in capturing complex relationships and improving classification accuracy, while standard PCA remains a reliable and interpretable option. The study emphasizes the benefits of PCA and kernel

PCA in data analysis and classification tasks, providing insights for researchers and practitioners in various domains.

Introduction

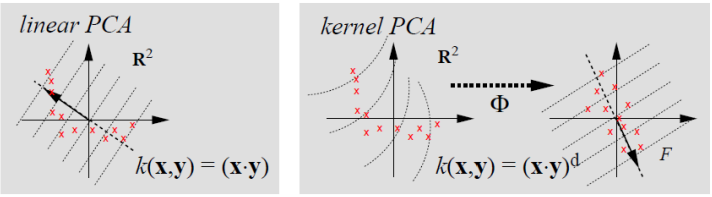

Figure 1: Baisc idea of kernel PCA [5]

PCA is a widely utilized method for data analysis and dimension reduction [3]. It facilitates the visualization of multidimensional data and enhances data interpretability while preserving the most relevant information. Standard PCA works effectively with data that can be distinctly separated into different categories using a straight line (in 2D) or a hyperplane (in 3D and higher dimensions). However, it falls short with linearly inseparable data where classes can only be distinguished by utilizing curved decision boundaries. In such cases, standard PCA underperforms. To address this limitation, kernel PCA was developed for non-linear PCA [5]. It incorporates a non-linear kernel function alongside PCA, enabling the execution of non-linear PCA by transforming input matrices using the kernel function. Kernel PCA allows the extraction of information that differs from what traditional linear approximations provide. The basic idea of kernel PCA is performing linear PCA, just like a PCA in the input space, but it operates in a high-dimensional feature space F, as illustrated in Figure 1. The contour lines of constant projections onto the principal eigenvector (drawn as an arrow) become non-linear in the input space due to the non-linear connection of F to the input space through Φ. A preimage of the eigenvector cannot be represented in the input space as it may not even exist. The absence of a requirement to explicitly map into F is a crucial aspect of kernel PCA. Instead, all computations are conducted using a kernel function (k) in the input space (R^2 in this case).

The advantage of kernel PCA becomes apparent when considering that N points are often challenging to linearly separate in dimensions $d < N$. However, they can almost always be linearly separated in $d \geq N$ dimensions, which underscores the importance of kernel PCA. In other words, by mapping N points x_i into a feature space with $\Phi(x_i) \in \mathbb{R}^N$, it becomes straightforward to create a hyperplane that clusters the points into distinct groups. To achieve this, we employ kernels rather than directly entering the feature space, using $K = k(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$. The standard steps of kernel PCA dimensionality reduction can be summarized as [6]:

- Construct the kernel matrix K from the data set $\{x_i\}$ using the kernel function.
- Compute the Gram matrix $\tilde{K}$ using $\tilde{K} = K - 1_N K - K 1_N + 1_N K 1_N$, where 1_N is the $N \times N$ matrix with all elements equal to $1/N$.
- Using $K a_k = \lambda_k N a_k$ to solve for the vectors a_i (substitute K with $\tilde{K}$), where a_k is the N-dimensional column vector of a_{ki} [1].
- Compute the kernel principal components $y_k(x)$ using $y_k(x) = \sum_{i=1}^{N} a_{ki} k(x, x_i)$.

The process of transforming linearly inseparable data into a higher-dimensional space where linear separation becomes possible is known as the kernel trick [2]. Kernels serve as data transformation functions to facilitate this process. In this project, we explore various kernels, including Gaussian RBF kernel, Polynomial kernel, Laplacian kernel, and Sigmoid kernel. Table 1 illustrates the corresponding kernel functions. The wine dataset is from the UCI Machine Learning Repository, which consists of the results of a chemical analysis of wines produced in the same region of Italy but derived from three different cultivars. It comprises 178 observations with 14 variables. Among these variables, 13 are continuous and represent the chemical analysis of wines, such as the quantity of flavonoids present. The remaining variable denotes the wine varieties, labeled from 1 to 3. The objective of this project is to investigate the application of

Kernel type	Function
Gaussian RBF kernel	$exp(-\|\|x_i - x_j\|\|^2/2\sigma^2)$
Polynomial kernel	$(x_i^T x_j + c)^d$
Laplacian kernel	$exp(-\sigma\|\|x_i - x_j\|\|_1)$
Sigmoid kernel	$tanh(\alpha x_i^T x_j + c)$

Table 1: Kernel function for each kernel PCA

standard PCA and kernel PCA for analyzing the wine dataset and determining the most suitable method for this specific dataset.

Methodology

The analyses for this report were conducted using **R** software, specifically version 4.3.0. To evaluate the performance of each PCA method, including standard PCA and various kernel PCA techniques, they were employed as preprocessing steps for the wine type classification task. To analyze the preprocessed data with multiple classes, the multinomial logistic regression model was employed. This model constructs a linear predictor function that combines a set of weights with the explanatory variables of each observation, generating a score [4]. We utilized the *multinom* function from the *nnet* package to implement this model. By establishing a relationship between the transformed features and the target variable (wine types), we can make predictions based on the transformed data. To ensure a fair comparison among the different PCA approaches, all variables were scaled before conducting the analysis. This scaling process involved adjusting the mean of each variable to zero and the standard deviation to one. Subsequently, both standard PCA and kernel PCA were applied to the scaled data to obtain transformed data for each method. For kernel PCA, appropriate kernel function selection and parameter tuning were performed. The transformed data was then randomly divided, allocating 70% to the training set and reserving the remaining 30% as the test set. For each method, the original

data for the training set and the test set are the same. The multinomial logistic regression model was fitted using the transformed data from the training set, enabling predictions to be made for the transformed data in the test set. By comparing the predicted results with the actual wine types in the test set, we determined the accuracy of each PCA technique. Accuracy serves as a metric to gauge the effectiveness of each PCA method, with higher accuracy indicating better performance. Through this evaluation process, we can identify the most suitable PCA approach for the wine dataset based on its ability to accurately capture the underlying patterns in the data.

Results

The classification accuracy and the corresponding number of principal components are presented in Table 2. Remarkably, all five methods achieved accuracy rates exceeding 95%, which is highly commendable. Among these approaches, the Sigmoid kernel PCA exhibited the highest accuracy, correctly predicting all test data. However, this method utilized a large number of principal components, specifically 73, which is considerably high. On the other hand, the remaining methods achieved an accuracy of 98.11%, except for the standard PCA. Notably, the Laplacian kernel PCA demonstrated the fewest number of principal components, with only 7. This constitutes approximately half the number of principal components required by the standard PCA. Although the standard PCA yielded slightly lower accuracy (96.23%), its performance remains highly satisfactory. Overall, the results showcase the effectiveness of the employed PCA methods, with excellent classification accuracy observed across the board. The Sigmoid kernel PCA stands out for its perfect prediction rate, albeit with a high number of principal components. Conversely, the Laplacian kernel PCA offers comparable accuracy while utilizing significantly fewer principal components. The standard PCA, despite having slightly lower accuracy, still performs admirably well (96.23%).

Method	Accuracy	Number of PCs
Standard PCA	96.23%	13
Gaussian RBF kernel PCA	98.11%	18
Polynomial kernel PCA	98.11%	104
Laplacian kernel PCA	98.11%	7
Sigmoid kernel PCA	100%	73

Table 2: Classification results for each method

Kernel PCA preserves the relative distances between data points, which can be beneficial for clustering purposes. Additionally, it allows for non-linear mappings of the data through the use of kernel functions, enabling the capture of complex relationships and structures that may go unnoticed with standard PCA. This superior classification performance of kernel PCAs was evident in the wine classification results. Figure 2 showcases the visualization results of both standard PCA and kernel PCA. It is important to note that kernel PCA operates in a high-dimensional feature space induced by the kernel function, making the interpretation of results more challenging compared to standard PCA. The transformed features are not directly associated with the original variables. Therefore, for the purpose of interpretation, standard PCA is preferred to identify the important variables in the wine dataset. Regarding the standard PCA visualization result, it appears that wine type 1 is fairly well classified when the PC2 score is below 0.8 and the PC1 score is less than 0. On the other hand, wine type 3 seems to be well classified when the PC1 score is positive and the PC2 score is negative. Additionally, when the PC2 score exceeds 0.8, wine type 2 appears to be well classified. To assess the variability accounted for by the principal components for standard PCA, Figure 3 presents the scree plot, while Table 3 demonstrates the proportion of variability accounted for by first three principal components. By employing the "eigenvalue greater than 1 rule" or the elbow rule, it is appropriate to consider three principal components. These three components account for approximately 66% of the total variability.

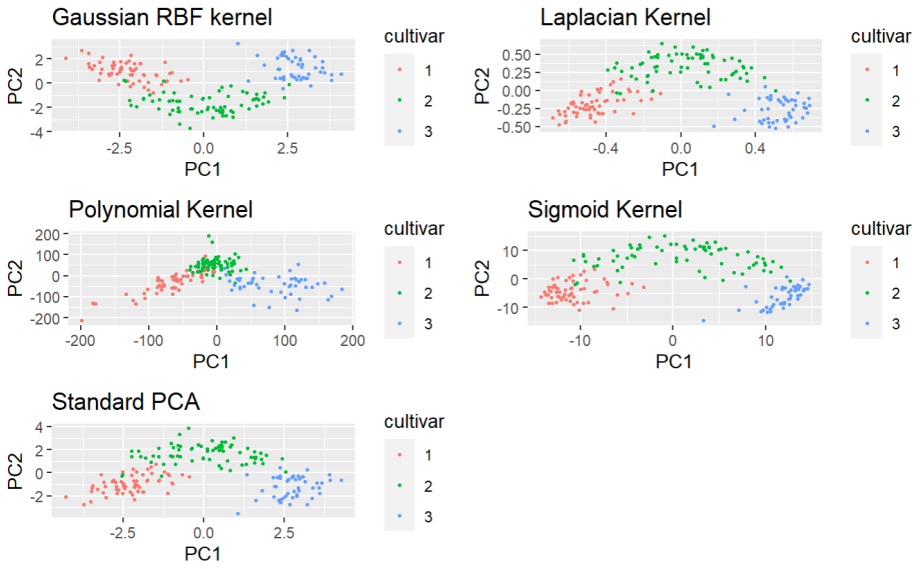

Figure 2: Visualization results for standard PCA and kernel PCA

Explained variance	PC1	PC2	PC3
%	36%	19%	11%

Table 3: Proportion of the variability by first 3 principal components

Figure 4 illustrates the correlations of PC1, PC2, and PC3 scores with the original variables. The amount of flavonoids in the wine exhibits a strong negative correlation (-0.92) with the PC1 score. This indicates that as the amount of flavonoids increases, the PC1 score tends to decrease. Similarly, the total phenolic content of the wine and the ratio of optical density at 280 nm to 315 nm in diluted wines also show strong negative correlations (-0.86 and -0.82, respectively) with the PC1 score. Hence, these variables play a similar role to flavonoids in determining the PC1 score. Other variables demonstrate moderate correlations with the PC1 score, ranging between 0.3 and 0.7 (or -0.7 and -0.3), except for the color intensity and ash content of the wine. In summary, wines with higher levels of flavonoids, total phenols, and a larger ratio of optical

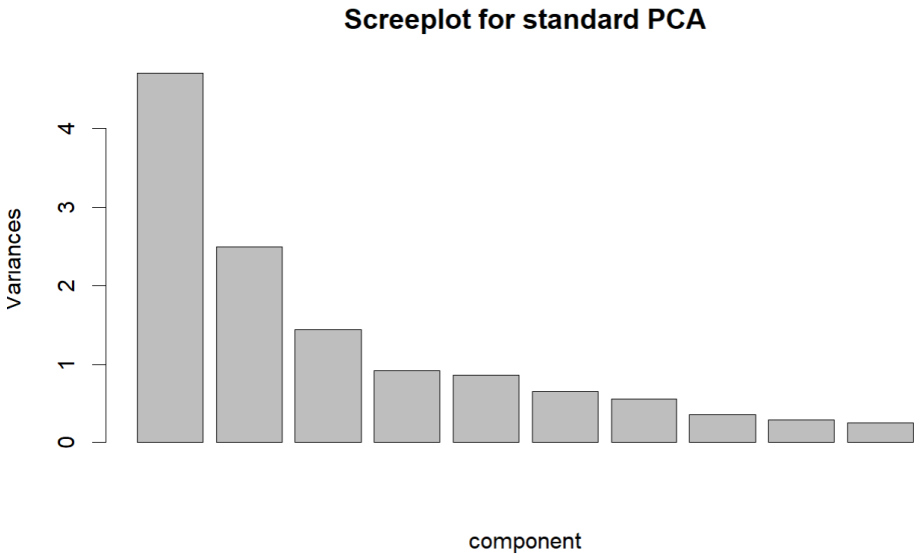

Figure 3: Screeplot for standard PCA

	Alcohol	Malic Acid	Ash	Alkalinity	Magnesium	Total phenols
PC1	-0.31	0.53	0	0.52	-0.31	-0.86
PC2	-0.76	-0.36	-0.50	0.02	-0.47	-0.10
PC3	-0.25	0.11	0.75	0.74	0.16	0.18

Flavonoids	Nonflavonoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315	Proline
-0.92	0.65	-0.68	0.19	-0.64	-0.82	-0.62
0.01	-0.05	-0.06	-0.84	0.44	0.26	-0.58
0.18	0.20	0.18	-0.17	0.10	0.20	-0.15

Figure 4: Correlations of PC1, PC2 and PC3 scores with the original variables

density tend to have smaller PC1 scores. Besides, the color intensity (-0.84) and alcohol content (-0.76) of the wine exhibit the strongest negative correlation with the PC2 score, while the ash content (0.75) and alkalinity (0.74) of the wine show the strongest positive correlation with the PC3 score. These variables that are strongly correlated with the first three components are all variables that are very important for the variation of the wine data set.

Discussion

This report explored the application of PCA and kernel PCA methods for analyzing and classifying a wine dataset. PCA proved to be a valuable technique for data analysis and dimension reduction, allowing for the visualization and interpretation of multidimensional data. However, when dealing with linearly inseparable data, kernel PCA emerged as a powerful alternative, enabling the capture of complex relationships through non-linear kernel functions. The results of this study showcased the effectiveness of both PCA and kernel PCA methods in classifying wine types. All tested methods achieved commendable accuracy rates, with the Sigmoid kernel PCA exhibiting the highest accuracy. Despite requiring a large number of principal components, the Sigmoid kernel PCA correctly predicted all test data. On the other hand, the remaining methods achieved similar accuracy rates, with the Laplacian kernel PCA requiring significantly fewer principal components. Kernel PCA's ability to preserve relative distances between data points and capture non-linear relationships proved advantageous for clustering and analyzing complex datasets. However, it is important to note that the interpretation of kernel PCA results can be more challenging due to the high-dimensional feature space induced by the kernel function. In contrast, standard PCA provided interpretable results directly associated with the original variables. The examination of correlations between principal component scores and the original variables shed light on the key factors contributing to the variability in the wine dataset. Variables such as flavonoids, total phenols, optical density ratios, color intensity, alcohol content,

ash content and alkalinity of the wine demonstrated strong correlations with specific principal components, indicating their importance in determining the dataset's variation.

Conclusion

In summary, this study demonstrated the effectiveness of PCA and kernel PCA methods for analyzing the wine dataset. The results emphasized the advantages of kernel PCA in capturing complex relationships and achieving high classification accuracy, while standard PCA remained a reliable and interpretable option. These findings contribute to the understanding of data analysis techniques and provide valuable insights for the analysis of similar datasets in various domains. Researchers and practitioners can leverage PCA and kernel PCA to enhance data interpretability and classification accuracy, ultimately improving decision-making processes in various fields.

References

[1] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[2] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

[3] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374:20150202, 2016.

[4] Scott Menard. *Applied logistic regression analysis*. Number 106 in Quantitative Applications in the Social Sciences. Sage, 2002.

[5] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[6] Quan Wang. Kernel principal component analysis and its applications in face recognition and active shape models. *arXiv preprint arXiv:1207.3538*, 2012.

Appendix

Listing 1: Data preprocessing

```
# read the data into R
df = read.csv("Wine.csv")
# convert the response variable to a factor
df$Customer_Segment = factor(df$Customer_Segment)
# have a glance at the data
str(df)
# scale the data
scale_df = data.frame(scale(df[,-14]))
# split the dataset into the training and test set
library(caTools)
# Set the seed for reproducibility
set.seed(66)
# specify the proportion of data to allocate for the training set
split = sample.split(df$Customer_Segment, SplitRatio = 0.7)
```

Listing 2: Standard PCA

```
# perform linear principal components analysis on the scaled data
linear_PCA = prcomp(scale_df)
# make the scree plot
screeplot(linear_PCA, main='Screeplot for standard PCA', xlab='component')
sum(linear_PCA$sdev^2)
# proportion of the variability
round(linear_PCA$sdev^2/sum(linear_PCA$sdev^2),2)
# compute the correlation of the first principal component score
# with the original variables
round(linear_PCA$rotation[,1] * linear_PCA$sdev[1],2)
# compute the correlation of the second principal component score
# with the original variables
```

```
round(linear_PCA$rotation[,2] * linear_PCA$sdev[2],2)
# compute the correlation of the third principal component score
# with the original variables
round(linear_PCA$rotation[,3] * linear_PCA$sdev[3],2)
# combine the principal component scores and add the label
linear_pca = cbind(data.frame(linear_PCA$x),cultivar=df$Customer_Segment)
# get the transformed training and test sets
training_set_linear = subset(linear_pca, split == TRUE)
test_set_linear = subset(linear_pca, split == FALSE)

# Setting the reference
training_set_linear$cultivar = relevel(training_set_linear$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_linear = multinom(cultivar ~ ., data = training_set_linear)
# Predicting the values for the test dataset
y_pred_linear = predict(model_linear, newdata = test_set_linear[,-14], "class")
#The Confusion Matrix
(tab_linear = table("real"=test_set_linear$cultivar, "pred" = y_pred_linear))
# Calculating accuracy - sum of diagonal elements divided by total obs
round((sum(diag(tab_linear))/sum(tab_linear))*100,2)
```

Listing 3: Gaussian RBF kernel PCA

```
# apply Kernel PCA
library(kernlab)

# tuning the parameters
sigma_values = seq(0.001, 2, by = 0.001)
best_accuracy = 0
best_sigma = 0

for (sigma in sigma_values) {
```

```
# Radial Basis kernel function "Gaussian"
G_PCA = kpca(~., data = scale_df,kernel="rbfdot",kpar=list(sigma=sigma))
# get the principal component scores
G_pca = as.data.frame(predict(G_PCA, scale_df))
# label the data
G_pca$cultivar = df$Customer_Segment
# get the transformed training and test sets
training_set_G = subset(G_pca, split == TRUE)
test_set_G = subset(G_pca, split == FALSE)

# Setting the reference
training_set_G$cultivar = relevel(training_set_G$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_G = multinom(cultivar ~ ., data = training_set_G)
# Predicting the values for the test dataset
y_pred_G = predict(model_G, newdata = test_set_G[,-ncol(training_set_G)],
                   "class")
#The Confusion Matrix
tab_G = table("real"=test_set_G$cultivar, "pred" = y_pred_G)
# Calculating accuracy - sum of diagonal elements divided by total obs
current_accuracy = round((sum(diag(tab_G))/sum(tab_G))*100,2)
# Keep track of the best scale, offset, and accuracy
    if (current_accuracy > best_accuracy) {
      best_accuracy = current_accuracy
      best_sigma = sigma
    }
  }
best_accuracy
best_sigma
```

```
# after tuning, select the approriate kernel function
# Radial Basis kernel function "Gaussian"
G_PCA = kpca(~., data = scale_df,kernel="rbfdot",kpar=list(sigma=0.003))
# get the principal component scores
G_pca = as.data.frame(predict(G_PCA, scale_df))
# label the data
G_pca$cultivar = df$Customer_Segment
# get the transformed training and test sets
training_set_G = subset(G_pca, split == TRUE)
test_set_G = subset(G_pca, split == FALSE)

# Setting the reference
training_set_G$cultivar = relevel(training_set_G$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_G = multinom(cultivar ~ ., data = training_set_G)
# Predicting the values for the test dataset
y_pred_G = predict(model_G, newdata = test_set_G[,-ncol(training_set_G)],
                   "class")
#The Confusion Matrix
(tab_G = table("real"=test_set_G$cultivar, "pred" = y_pred_G))
# Calculating accuracy - sum of diagonal elements divided by total obs
round((sum(diag(tab_G))/sum(tab_G))*100,2)
# number of principal components
ncol(training_set_G) - 1
```

Listing 4: Polynomial Kernel PCA

```
# tuning the parameter
degree_values = 2:10
best_accuracy = 0
```

```
best_degree = 0

for (degree in degree_values) {

# Polynomial kernel function
P_PCA = kpca(~., data = scale_df,kernel="polydot",kpar=list(degree=degree))
# get the principal component scores
P_pca = as.data.frame(predict(P_PCA, scale_df))
# label the data
P_pca$cultivar = df$Customer_Segment
# get the transformed training and test sets
training_set_P = subset(P_pca, split == TRUE)
test_set_P = subset(P_pca, split == FALSE)

# Setting the reference
training_set_P$cultivar = relevel(training_set_P$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_P = multinom(cultivar ~ ., data = training_set_P)
# Predicting the values for the test dataset
y_pred_P = predict(model_P, newdata = test_set_P[,-ncol(training_set_P)],
                   "class")
#The Confusion Matrix
tab_P = table("real"=test_set_P$cultivar, "pred" = y_pred_P)
# Calculating accuracy - sum of diagonal elements divided by total obs
current_accuracy = round((sum(diag(tab_P))/sum(tab_P))*100,2)

 # Keep track of the best scale, offset, and accuracy
    if (current_accuracy > best_accuracy) {
      best_accuracy = current_accuracy
      best_degree = degree
```

```
    }
  }
best_accuracy
best_degree

# after tuning, select the approriate kernel function
# Polynomial kernel function
P_PCA = kpca(~., data = scale_df,kernel="polydot",kpar=list(degree=2))
# get the principal component scores
P_pca = as.data.frame(predict(P_PCA, scale_df))
# label the data
P_pca$cultivar = df$Customer_Segment
# get the transformed training and test sets
training_set_P = subset(P_pca, split == TRUE)
test_set_P = subset(P_pca, split == FALSE)

# Setting the reference
training_set_P$cultivar = relevel(training_set_P$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_P = multinom(cultivar ~ ., data = training_set_P)
# Predicting the values for the test dataset
y_pred_P = predict(model_P, newdata = test_set_P[,-ncol(training_set_P)],
                   "class")
#The Confusion Matrix
(tab_P = table("real"=test_set_P$cultivar, "pred" = y_pred_P))
# Calculating accuracy - sum of diagonal elements divided by total obs
round((sum(diag(tab_P))/sum(tab_P))*100,2)
# number of principal components
ncol(training_set_P) - 1
```

Listing 5: Laplacian Kernel PCA

```
# tuning the parameter
sigma_values = seq(0.001, 1, by = 0.001)
best_accuracy = 0
best_sigma = 0

for (sigma in sigma_values) {

# Laplacian kernel function
L_PCA = kpca(~., data = scale_df,kernel="laplacedot",kpar=list(sigma = sigma))
# get the principal component scores
L_pca = as.data.frame(rotated(L_PCA))
# label the data
L_pca$cultivar = df$Customer_Segment
# get the transformed training and test sets
training_set_L = subset(L_pca, split == TRUE)
test_set_L = subset(L_pca, split == FALSE)

# Setting the reference
training_set_L$cultivar = relevel(training_set_L$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_L = multinom(cultivar ~ ., data = training_set_L)
# Predicting the values for the test dataset
y_pred_L = predict(model_L, newdata = test_set_L[,-ncol(training_set_L)],
                   "class")
#The Confusion Matrix
tab_L = table("real"=test_set_L$cultivar, "pred" = y_pred_L)
# Calculating accuracy - sum of diagonal elements divided by total obs
current_accuracy = round((sum(diag(tab_L))/sum(tab_L))*100,2)
```

```
# Keep track of the best scale, offset, and accuracy
    if (current_accuracy > best_accuracy) {
      best_accuracy = current_accuracy
      best_sigma = sigma
    }
  }

best_accuracy
best_sigma

# after tuning, select the approriate kernel function
# Laplacian kernel function
L_PCA = kpca(~., data = scale_df,kernel="laplacedot",kpar=list(sigma = 0.001))
# get the principal component scores
L_pca = as.data.frame(rotated(L_PCA))
# label the data
L_pca$cultivar = df$Customer_Segment
# get the transformed training and test sets
training_set_L = subset(L_pca, split == TRUE)
test_set_L = subset(L_pca, split == FALSE)

# Setting the reference
training_set_L$cultivar = relevel(training_set_L$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_L = multinom(cultivar ~ ., data = training_set_L)
# Predicting the values for the test dataset
y_pred_L = predict(model_L, newdata = test_set_L[,-ncol(training_set_L)],
                   "class")
#The Confusion Matrix
```

```
(tab_L = table("real"=test_set_L$cultivar, "pred" = y_pred_L))
# Calculating accuracy - sum of diagonal elements divided by total obs
round((sum(diag(tab_L))/sum(tab_L))*100,2)
# number of principal components
ncol(training_set_L) - 1
```

Listing 6: Sigmoid Kernel PCA

```
# tuning parameters
scales = seq(0.1, 1, by = 0.1)
offsets = seq(-1, 1, by = 0.1)

best_accuracy = 0
best_scale = 0
best_offset = 0

for (scale in scales) {
  for (offset in offsets) {

    # Perform Sigmoid Kernel PCA
    S_PCA = kpca(~., data=scale_df,kernel="tanhdot",
                 kpar=list(scale=scale,offset=offset))
    # get the principal component scores
    S_pca = as.data.frame(rotated(S_PCA))
    # label the data
    S_pca$cultivar = df$Customer_Segment
    # get the transformed training and test sets
    training_set_S = subset(S_pca, split == TRUE)
    test_set_S = subset(S_pca, split == FALSE)

    # Setting the reference
```

```
    training_set_S$cultivar = relevel(training_set_S$cultivar, ref = "1")
    #fitting training data to the Multinomial Regression Model
    model_S = multinom(cultivar ~ ., data = training_set_S)
    # Predicting the values for the test dataset
    y_pred_S = predict(model_S, newdata = test_set_S[,-ncol(training_set_S)],
                       "class")
  #The Confusion Matrix
  tab_S = table("real"=test_set_S$cultivar, "pred" = y_pred_S)
  # Calculating accuracy - sum of diagonal elements divided by total obs
  current_accuracy  = round((sum(diag(tab_S))/sum(tab_S))*100,2)

    # Keep track of the best scale, offset, and accuracy
    if (current_accuracy > best_accuracy) {
      best_accuracy = current_accuracy
      best_scale = scale
      best_offset = offset
    }
  }
}

best_accuracy
best_scale
best_offset

# after tuning, select the approriate kernel function
# Perform Sigmoid Kernel PCA
S_PCA = kpca(~., data=scale_df,kernel="tanhdot",kpar=list(scale=0.2,offset=0.1))
# get the principal component scores
S_pca = as.data.frame(rotated(S_PCA))
# label the data
```

```
S_pca$cultivar = df$Customer_Segment
# get the transformed training and test sets
training_set_S = subset(S_pca, split == TRUE)
test_set_S = subset(S_pca, split == FALSE)

# Setting the reference
training_set_S$cultivar = relevel(training_set_S$cultivar, ref = "1")
#fitting training data to the Multinomial Regression Model
model_S = multinom(cultivar ~ ., data = training_set_S)
# Predicting the values for the test dataset
# Predicting the values for the test dataset
y_pred_S = predict(model_S, newdata = test_set_S[,-ncol(training_set_S)],
                   "class")
#The Confusion Matrix
(tab_S = table("real"=test_set_S$cultivar, "pred" = y_pred_S))
# Calculating accuracy - sum of diagonal elements divided by total obs
round((sum(diag(tab_S))/sum(tab_S))*100,2)
# number of principal components
ncol(training_set_S) - 1
```

Listing 7: Visualization results

```
# plot
SP = data.frame(X1=linear_PCA$x[,1],X2=linear_PCA$x[,2],
                 kernel=rep("Standard PCA",178),cultivar = df$Customer_Segment)
G = data.frame(rotated(G_PCA)[,1:2],kernel=rep("Gaussian RBF kernel",178),
                cultivar = df$Customer_Segment)
P = data.frame(rotated(P_PCA)[,1:2],kernel=rep("Polynomial Kernel",178),
                cultivar = df$Customer_Segment)
L = data.frame(rotated(L_PCA)[,1:2],kernel=rep("Laplacian Kernel",178),
                cultivar = df$Customer_Segment)
S = data.frame(rotated(S_PCA)[,1:2],kernel=rep("Sigmoid Kernel",178),
```

```
                cultivar = df$Customer_Segment)
ALL = rbind(SP,G,P,L,S)

split_data = split(ALL, ALL$kernel)

library(ggplot2)
plot_list = lapply(names(split_data), function(kernel) {
  ggplot(split_data[[kernel]], aes(x = X1, y = X2,color=cultivar)) +
    geom_point(size=.5) +
    labs(x = "PC1", y = "PC2",title=kernel)
})
# Arrange the scatter plots in a grid
gridExtra::grid.arrange(grobs = plot_list, nrow = 3)
```