Machine Learning

Principal Component Analysis

Ayşe Ceren Çiçek

PCA is an unsupervised algorithm that is used for dimensionality reduction.

Its goal is to identify and detect the correlation between variables. It is attempting to learn about relationship between values by finding a list of principal axes and it projects high dimensional data into smaller dimensional subspace while keeping most of the information.

PCA's key advantages are:

- its low noise sensitivity
- the decreased requirements for capacity and memory
- increased efficiency has given the processes taking place in smaller dimensions

PCA is highly get affected by outliers.

Dataset: https://archive.ics.uci.edu/ml/datasets/wine

The wine dataset contains the results of a chemical analysis of wines grown in a specific area of Italy. Three types of wine are represented in the 178 samples, with the results of 13 chemical analyses recorded for each sample.

The columns are:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline
- Custome Segment

Using pca we will try to reduce the dimensionality of this dataset. It contains 13 dimensions(13 independent variables). We will end up with new two independent variables that are among those original 13 independent variables. Our dependent variable is Customer_Segment.

Importing libraries

```
#install.packages('caTools')
#install.packages('caret')
#install.packages('e1071')
library(caTools)
library(caret)
```

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

```
## Loading required package: lattice
```

```
library(e1071)
library(magrittr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Loading dataset

```
dataset = read.csv('dataset.csv')
head(dataset, n=5)
```

```
##   Alcohol Malic_Acid  Ash Ash_Alcanity Magnesium Total_Phenols Flavanoids
## 1   14.23       1.71 2.43         15.6       127          2.80       3.06
## 2   13.20       1.78 2.14         11.2       100          2.65       2.76
## 3   13.16       2.36 2.67         18.6       101          2.80       3.24
## 4   14.37       1.95 2.50         16.8       113          3.85       3.49
## 5   13.24       2.59 2.87         21.0       118          2.80       2.69
##   Nonflavanoid_Phenols Proanthocyanins Color_Intensity  Hue OD280 Proline
## 1                 0.28            2.29            5.64 1.04  3.92    1065
## 2                 0.26            1.28            4.38 1.05  3.40    1050
## 3                 0.30            2.81            5.68 1.03  3.17    1185
## 4                 0.24            2.18            7.80 0.86  3.45    1480
## 5                 0.39            1.82            4.32 1.04  2.93     735
##   Customer_Segment
## 1                1
## 2                1
## 3                1
## 4                1
## 5                1
```

Split data into Train and Test set

Customer_Segment column is our dependent variable.

```
set.seed(123)
splitted = sample.split(dataset$Customer_Segment, SplitRatio = 0.8)
trainSet = subset(dataset, splitted == TRUE)
testSet = subset(dataset, splitted == FALSE)
```

Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data.

We will scale all the features except the last one Customer_Segment

```
trainSet[-14] = scale(trainSet[-14])
testSet[-14] = scale(testSet[-14])
```

Apply Principal Component Analysis

If we want our new extracted features to explain at least 60 percent of the variance, we need to specify it on thresh parameter. It is a cutoff for the cumulative percent of variance to be retained by PCA.

pcaComp parameter gets the specific number of PCA components to keep. For our case, from 13 variables we will only keep 2. If this parameter is specified, this over-rides thresh. That's why we will not specify the thresh parameter.

Because PCA is unsupervised learning it does not consider the dependent variable. So we will remove the dependent variable which is the last column Customer_Segment from the training set.

```
pca = preProcess(x = trainSet[-14], method = 'pca', pcaComp = 2)
trainSet = predict(pca, trainSet)
testSet = predict(pca, testSet)

# 5 lines of the training set
head(trainSet, n=5)
```

```
##   Customer_Segment       PC1        PC2
## 1                1 -3.249569  1.5661160
## 2                1 -2.165889 -0.3186768
## 3                1 -2.501192  1.2353892
## 6                1 -2.941040  2.2999654
## 7                1 -2.393131  1.3228050
```

This is our training set. With PCA we reduced the number of dimensions from 13 to 2. PC1 and PC2 are our new dimensions.

Now we will move the Customer_Segment column to end of the training set and test set.

```
# 2 -> PC1, 3 -> PC2, 1 -> Customer_Segment
trainSet = trainSet[c(2, 3, 1)]
testSet = testSet[c(2, 3, 1)]

head(trainSet, n=5)
```

```
##         PC1        PC2 Customer_Segment
## 1 -3.249569  1.5661160                1
## 2 -2.165889 -0.3186768                1
## 3 -2.501192  1.2353892                1
## 6 -2.941040  2.2999654                1
## 7 -2.393131  1.3228050                1
```

We will build a classification model.

Fit Support Vector Machine model to training set

```
model = svm(formula = Customer_Segment ~ ., data = trainSet, type = 'C-classification', kernel = 'linea
```

Predict test results

```
# -3 to remove dependent variable
prediction = predict(model, newdata = testSet[-3])
prediction
```

```
##   4   5   8  11  16  20  21  24  31  32  50  59  65  67  68  69  87  88  89 104
##   1   1   1   1   1   1   1   1   1   1   1   1   2   2   2   2   2   2   2   2
## 106 107 111 114 118 126 132 134 137 138 139 145 151 167 173 174
##   2   2   2   2   2   2   3   3   3   3   3   3   3   3   3   3
## Levels: 1 2 3
```

Confusion Matrix

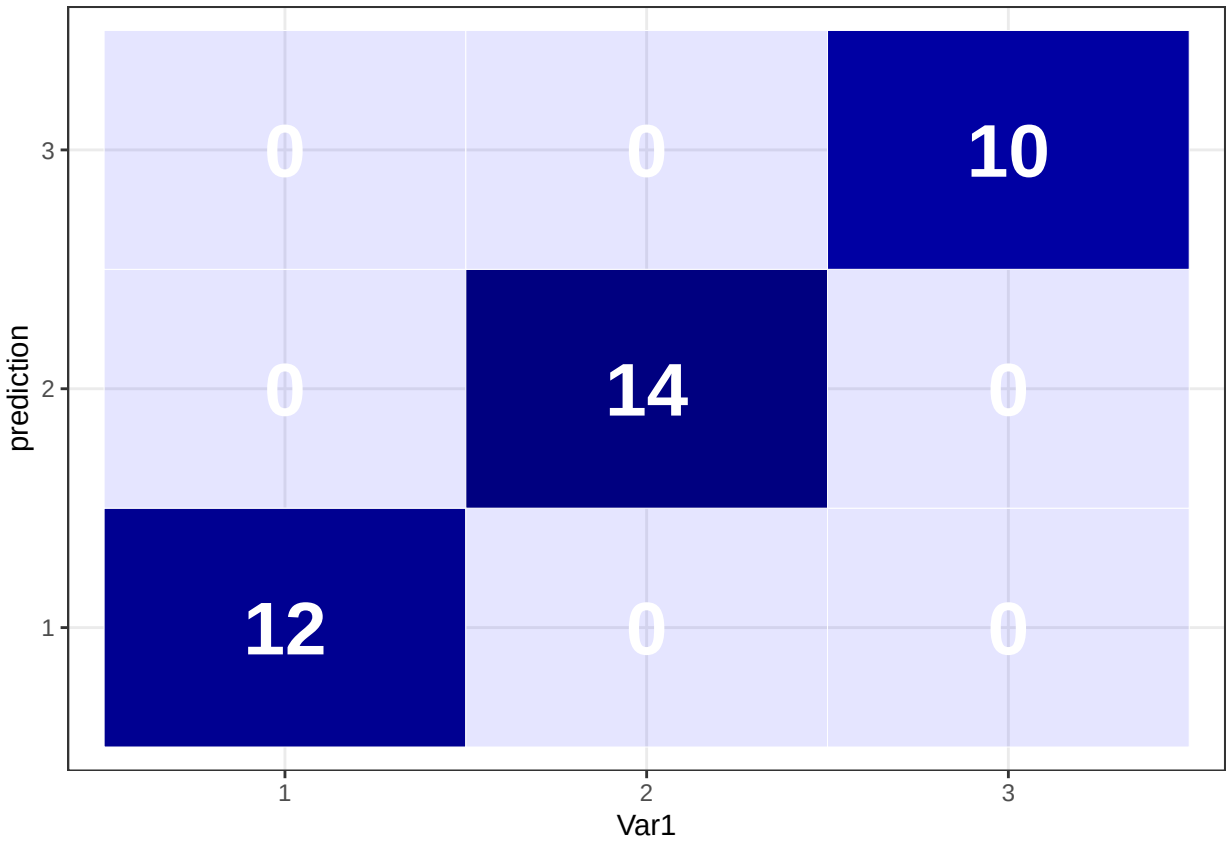
```
# Generate confusion matrix
conf.matrix <- confusionMatrix(table(testSet[, 3], prediction))
conf.matrix
```

```
## Confusion Matrix and Statistics
##
##    prediction
##      1  2  3
##   1 12  0  0
##   2  0 14  0
##   3  0  0 10
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9026, 1)
##     No Information Rate : 0.3889
##     P-Value [Acc > NIR] : 1.713e-15
##
##                   Kappa : 1
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000
## Prevalence             0.3333   0.3889   0.2778
## Detection Rate         0.3333   0.3889   0.2778
## Detection Prevalence   0.3333   0.3889   0.2778
## Balanced Accuracy      1.0000   1.0000   1.0000
```

As seen above, all of the predictions are correct and the accuracy is 1.

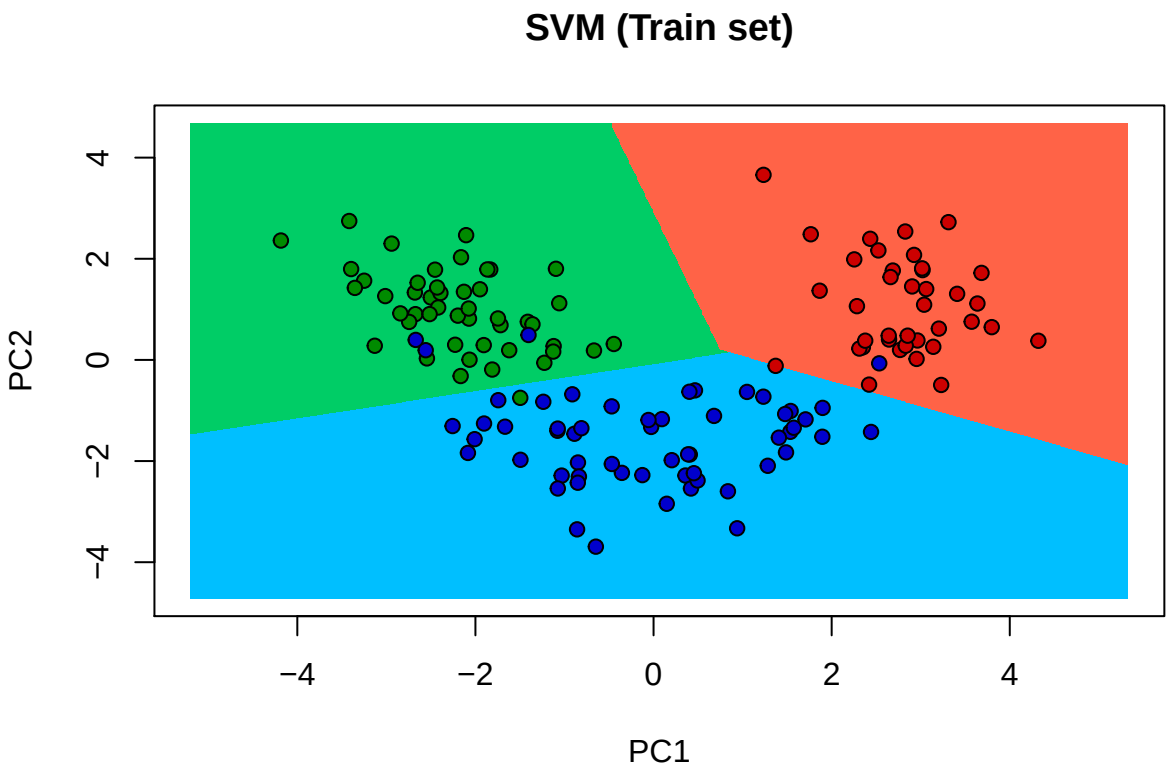
```
# Heatmap visualization of confusion matrix
table <- data.frame(conf.matrix$table)


plotTable <- table %>%
  group_by(prediction) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data =  plotTable, mapping = aes(x = Var1, y = prediction, alpha = prop)) +
  geom_tile(aes(fill = Freq), colour = "white") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1, color="white", size=10) +
  scale_fill_gradient(low = "blue", high = "navyblue") +
  theme_bw() + theme(legend.position = "none")
```



Visualize Train Set Results

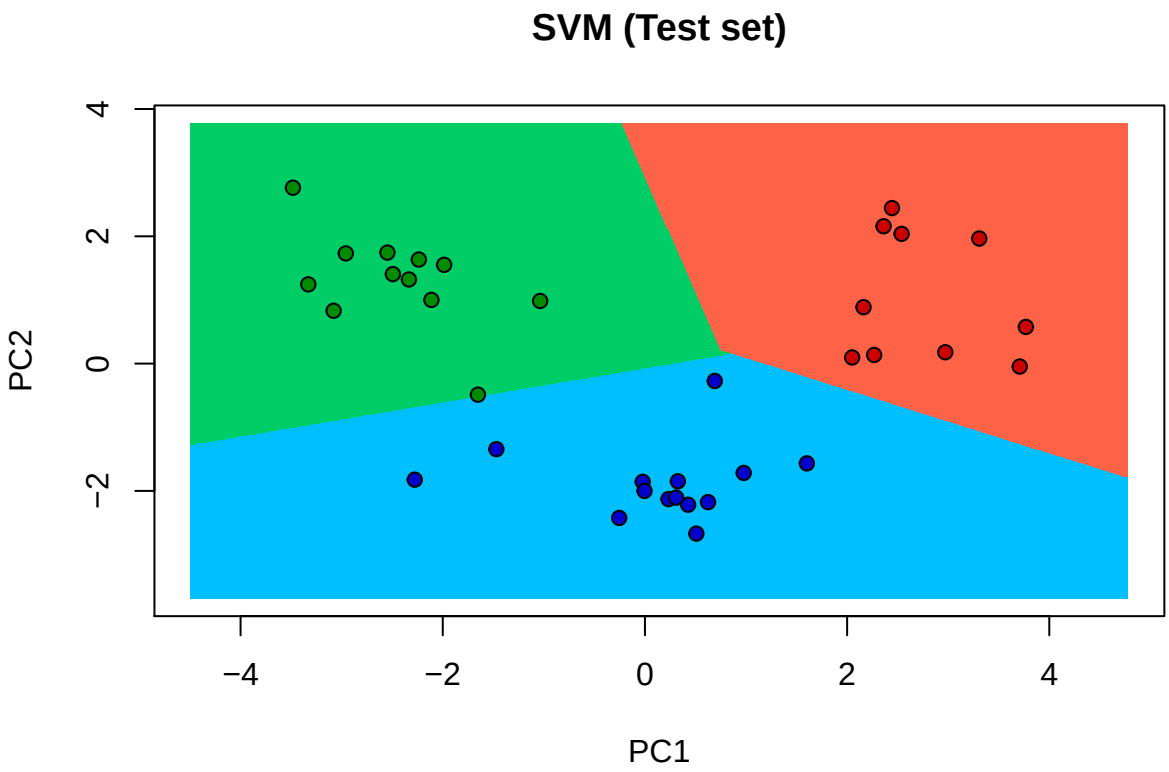
We have 3 dimensions with our dependent variable.

```
set = trainSet
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)

colnames(grid_set) = c('PC1', 'PC2')

y_grid = predict(model, newdata = grid_set)

plot(set[, -3], main = 'SVM (Train set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

points(grid_set, pch = '.', col = ifelse(y_grid == 2, 'deepskyblue', ifelse(y_grid == 1, 'springgreen3'
points(set, pch = 21, bg = ifelse(set[, 3] == 2, 'blue3', ifelse(set[, 3] == 1, 'green4', 'red3')))
```



Visualize Tes Set Results

```
set = testSet
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)

colnames(grid_set) = c('PC1', 'PC2')

y_grid = predict(model, newdata = grid_set)

plot(set[, -3], main = 'SVM (Test set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

points(grid_set, pch = '.', col = ifelse(y_grid == 2, 'deepskyblue', ifelse(y_grid == 1, 'springgreen3'
points(set, pch = 21, bg = ifelse(set[, 3] == 2, 'blue3', ifelse(set[, 3] == 1, 'green4', 'red3')))
```



All points are on their own regions. For example, all green points are on the green region. All of the predictions are correct.