Principal Component Analysis

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```
#install.packages("jsonlite", repos="https://cran.rstudio.com/")
library("jsonlite")

json_file <- 'https://datahub.io/machine-learning/vehicle/datapackage.json'
json_data <- fromJSON(paste(readLines(json_file), collapse=""))

## Warning in readLines(json_file): incomplete final line found on 'https://
## datahub.io/machine-learning/vehicle/datapackage.json'

# get list of all resources:
print(json_data$resources$name)

## [1] "validation_report" "vehicle_csv"       "vehicle_json"
## [4] "vehicle_zip"       "vehicle_arff"      "vehicle"

# print all tabular data(if exists any)
for(i in 1:length(json_data$resources$datahub$type)){
  if(json_data$resources$datahub$type[i]=='derived/csv'){
    path_to_file = json_data$resources$path[i]
    data <- read.csv(url(path_to_file))
    #print(data)
  }
}
```

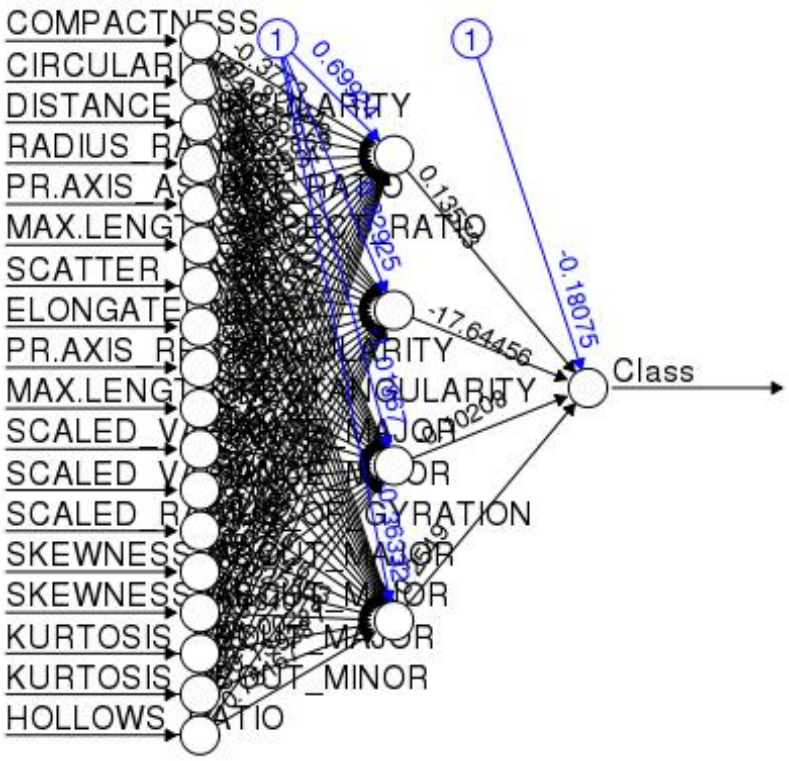
Keep on 'Saab' and 'bus' classes

```
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

dataset = data %>% filter(Class %in% c('saab','bus')) %>% transform(Class =
ifelse(Class=="saab",0,1))
```

```
dataset = as.data.frame(sapply(dataset, as.numeric))
```

split into training and testing

```
# Spliting training and testing dataset
index = sample( 1:nrow( dataset ), nrow( dataset ) * 0.6, replace = FALSE )

trainset = dataset[ index, ]
test = dataset[ -index, ]
target = 'Class'
testsetcols = setdiff(names(test),target)
testset = test[,testsetcols]

# Building a neural network (NN)
library( neuralnet )

##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute

n = names( trainset )
f = as.formula( paste( "Class ~", paste( n[!n %in% "Class"], collapse = "+" )
) )
nn = neuralnet( f, trainset, hidden = 4, linear.output = FALSE, threshold =
0.01 )

plot( nn, rep = "best" )
```




```
nn.results = compute( nn, testset )

results = data.frame( actual = test$Class, prediction = round(
nn.results$net.result ) )

library( caret )

## Loading required package: lattice

## Loading required package: ggplot2

t = table( results )
print( confusionMatrix( t ) )

## Confusion Matrix and Statistics
##
##       prediction
## actual  0  1
##      0  0 93
##      1  2 79
##
##                Accuracy : 0.454023
##                  95% CI : (0.3785258, 0.5311133)
##     No Information Rate : 0.9885057
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0230227
```

```
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.00000000
##             Specificity : 0.45930233
##          Pos Pred Value : 0.00000000
##          Neg Pred Value : 0.97530864
##              Prevalence : 0.01149425
##          Detection Rate : 0.00000000
##    Detection Prevalence : 0.53448276
##       Balanced Accuracy : 0.22965116
##
##        'Positive' Class : 0
##
```

Results without PCA

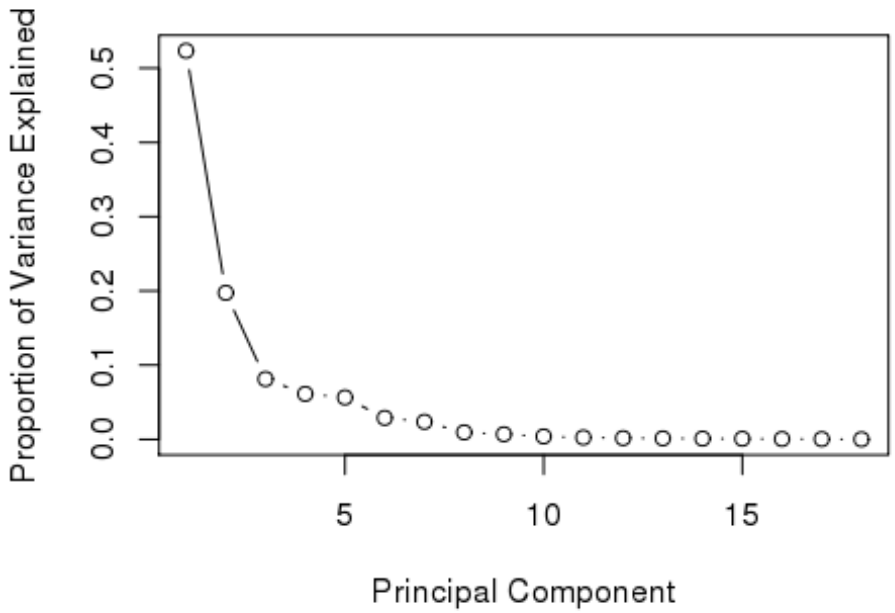
It seems we got some results. Firstly, take a look at the confusion matrix. Basically, confusion matrix says how much examples were classified into classes. The main diagonal shows the examples that were classify correctly and secondary diagonal shows misclassification. In this first result, the classifier shows itself very confused, because it classified correctly almost all examples from "saab" class, but it also classified most examples of "bus" class as "saab" class. Reinforcing this results, we can see that the value of accuracy is around 50%, it is a really bad result for classification task. The classifier has essentially a probability of 50% to classify a new example into "car" class and 50% into "bus" classes. Analogically, the neural network is tossing a coin for each new example to choose which class it should classify it in.

Let's see if PCA can help us

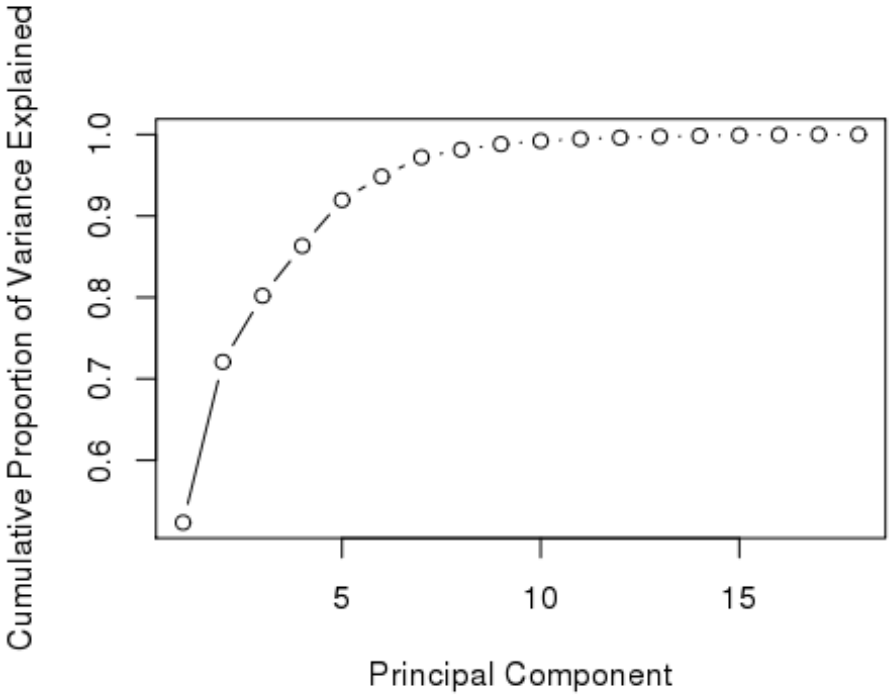
Now, let's perform the principal component analysis over the dataset and get the eigenvalues and eigenvectors. In a practical way, you will see that PCA function from R package provides a set of eigenvalues already sorted in descending order, it means that the first component is that one with the highest variance, the second component is the eigenvector with the second highest variance and so on. The code below shows how to chose the eigenvectors looking at the eigenvalues.

```
# PCA
pca_trainset = trainset[,testsetcols]
pca_testset = testset
pca = prcomp( pca_trainset, scale = T )

# variance
pr_var = ( pca$sdev )^2

# % of variance
prop_varex = pr_var / sum( pr_var )

# Plot
```

```
plot( prop_varex, xlab = "Principal Component",
                  ylab = "Proportion of Variance Explained", type = "b" )
```




```
# Scree Plot
plot( cumsum( prop_varex ), xlab = "Principal Component",
                            ylab = "Cumulative Proportion of Variance
Explained", type = "b" )
```

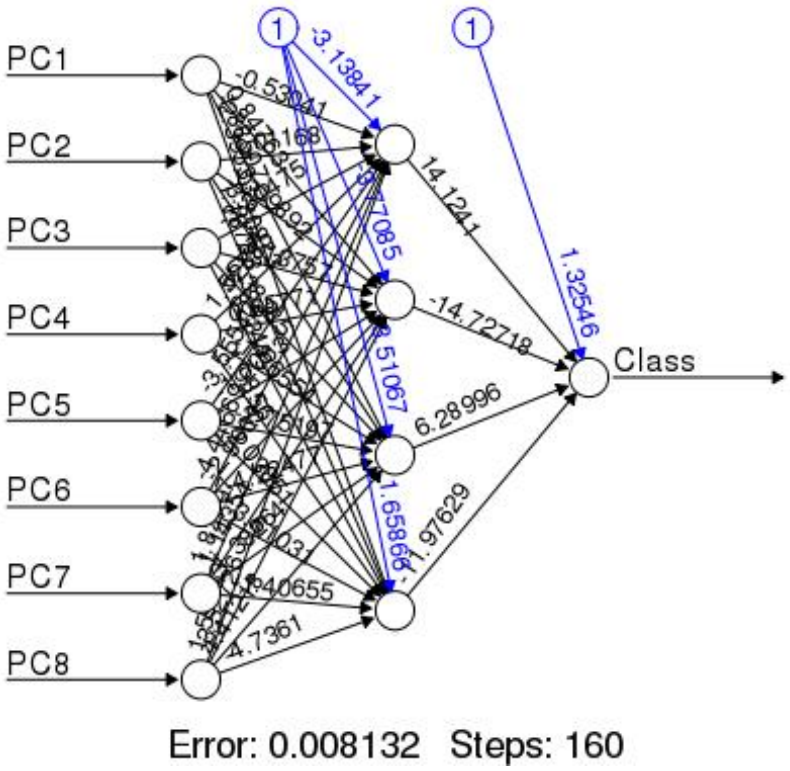


The native R function “prcomp” from stats default packages performs PCA, it returns all eigenvalues and eigenvectors needed. The first plot shows the percentage of variance of each feature. You can see that the first component has the highest variance, something value around 50% while the 8th component is around of 0% of variance. So, it indicates that we should pick up the first eight components. The second figure show an another perspective of the variance, though the cumulative sum over all the variance, you can see that the first eight eigenvalues correspond to approximately 98% of the all variance. Indeed, it is a pretty good number, it means that there is just 2% of information being lost. The biggest beneficial is that we are moving from a space with eighteen features to another one with only eight feature losing only 2% of information. That is the power of dimensionality reduction, definitively.

Now that we know the number of the features that will compose the new space, let’s create the new dataset and then model the neural network again and check if we get new better outcomes.

```
# Creating a new dataset
train = data.frame( Class = trainset$Class, pca$x )
t = as.data.frame( predict( pca, newdata = pca_testset ) )

new_trainset = train[, 1:9]
new_testset =  t[, 1:8]

# Build the neural network (NN)
library( neuralnet )
n = names( new_trainset )
```

```
f = as.formula( paste( "Class ~", paste( n[!n %in% "Class" ], collapse = "+"
) ) )
nn = neuralnet( f, new_trainset, hidden = 4, linear.output = FALSE,
threshold=0.01 )

# Plot the NN
plot( nn, rep = "best" )
```




```
# Test the resulting output
nn.results = compute( nn, new_testset )

# Results
results = data.frame( actual = test$Class,
                      prediction = round( nn.results$net.result ) )

# Confusion Matrix
library( caret )
t = table( results )
print( confusionMatrix( t ) )

## Confusion Matrix and Statistics
##
##       prediction
## actual  0  1
##      0 91  2
##      1  0 81
##
```

```
##                Accuracy : 0.9885057
##                  95% CI : (0.9590975, 0.9986049)
##     No Information Rate : 0.5229885
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.9769384
##  Mcnemar's Test P-Value : 0.4795001
##
##             Sensitivity : 1.0000000
##             Specificity : 0.9759036
##          Pos Pred Value : 0.9784946
##          Neg Pred Value : 1.0000000
##              Prevalence : 0.5229885
##          Detection Rate : 0.5229885
##    Detection Prevalence : 0.5344828
##       Balanced Accuracy : 0.9879518
##
##        'Positive' Class : 0
##
```

Conclusion

Dimensionality Reduction plays a really important role in machine learning, especially when you are working with thousands of features. Principal Components Analysis are one of the top dimensionality reduction algorithm, it is not hard to understand and use it in real projects. This technique, in addition to making the work of feature manipulation easier, it still helps to improve the results of the classifier, as we saw in this post.

Finally, the answer of the initial questioning is yes, indeed Principal Component Analysis helps improve the outcome of a classifier.