Practical No.3

AIM: Practical of Principal Component Analysis(PCA).

Theory:

PCA is a very popular method of dimensionality reduction because it provides a way to easily reduce the dimensions and is easy to understand. For this reason, PCA has been used in various applications from image compression to complex gene comparison. While using PCA, one should keep in mind its limitations well.

PCA is very sensitive to the scale of the data. It will create an initial basis in the direction of the largest variance in the data. Moreover, PCA applies a transformation over the data where all new components are orthogonal. The new features may not be interpretable in business.

Another limitation of PCA is the reliance on the mean and variance of the data. If the data has a relationship in higher dimensions such as kurtosis and skewness then PCA may not be the right technique to use on the data. In situations when the features are already orthogonal to each other and are uncorrelated, PCA will not produce any useful results except ordering the features in decreasing order of their variances.

PCA is very useful in situations when the data at hand is very large. Example, in case of image compression, PCA can be used to store the image in the first few hundred components and use less number of pixels.

We can implement the same in R programming language.

The **princomp()** function in R calculates the principal components of any data. We will also compare our results by calculating eigenvectors and eigenvalues separately. Let’s use the **IRIS dataset.**

Let’s start by loading the dataset.

```
# Taking the numeric part of the IRIS data

> data_iris <- iris[1:4]
```

The iris dataset having 150 observations (rows) with 4 features.
Let’s use the **cov()** function to calculate the covariance matrix of the loaded iris data set.

```
# Calculating the covariance matrix
> Cov_data <- cov(data_iris )
```

The next step is to calculate the eigenvalues and eigenvectors.
We can use the **eigen()** function to do this automatically for us.

```
# Find out the eigenvectors and eigenvalues using the covariance matrix
> Eigen_data <- eigen(Cov_data)
```

We have calculated the Eigen values from the data. We will now look at the PCA function **princomp()** which automatically calculates these values.
Let’s calculate the components and compare the values.

```
# Using the inbuilt function
```

```
> PCA_data <- princomp(data_iris ,cor="False")

# Let’s now compare the output variances
> Eigen_data$values
```
Output:
```
[1] 4.22824171 0.24267075 0.07820950 0.02383509

> PCA_data$sdev^2
   Comp.1     Comp.2     Comp.3     Comp.4
4.20005343 0.24105294 0.07768810 0.02367619
```

There is a slight difference due to squaring in PCA_data but the outputs are more or less simil ar. We can also compare the eigenvectors of both models.

```
> PCA_data$loadings[,1:4]
                  Comp.1      Comp.2      Comp.3     Comp.4
Sepal.Length  0.36138659  0.65658877  0.58202985  0.3154872
Sepal.Width  -0.08452251  0.73016143 -0.59791083 -0.3197231
Petal.Length  0.85667061 -0.17337266 -0.07623608 -0.4798390
Petal.Width   0.35828920 -0.07548102 -0.54583143  0.7536574

> Eigen_data$vectors
            [,1]        [,2]        [,3]       [,4]
[1,]  0.36138659 -0.65658877 -0.58202985  0.3154872
[2,] -0.08452251 -0.73016143  0.59791083 -0.3197231
[3,]  0.85667061  0.17337266  0.07623608 -0.4798390
[4,]  0.35828920  0.07548102  0.54583143  0.7536574
```

This time the eigenvectors calculated are same and there is no difference.

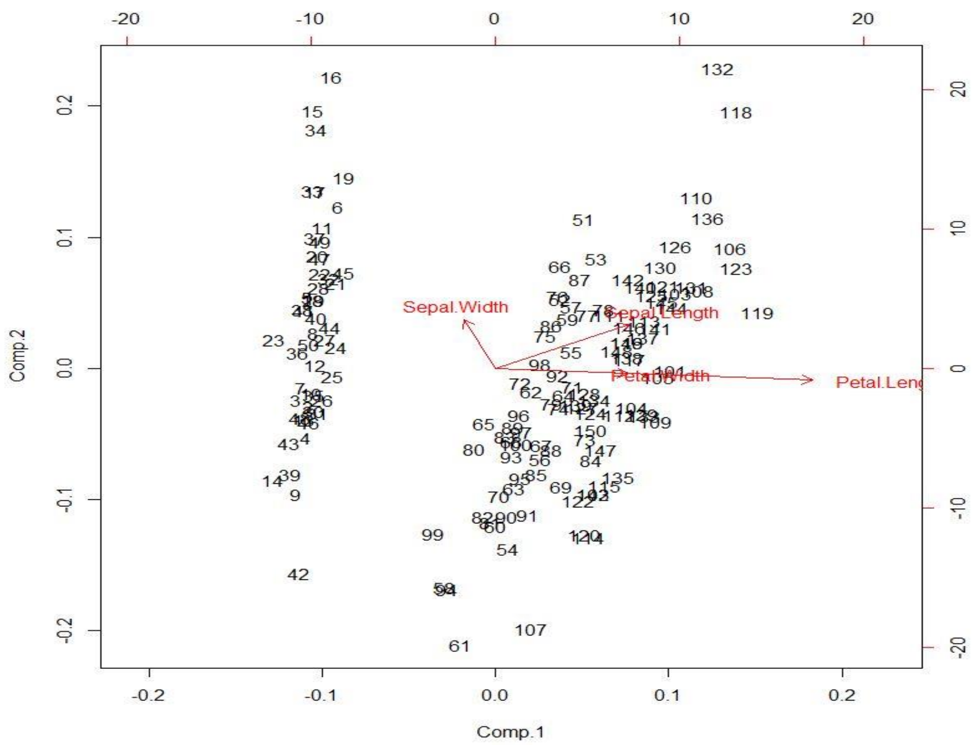
Let us now understand our model. We transformed our 4 features into 4 new orthogonal components. To know the importance of the first component, we can view the summary of the model.

```
> summary(PCA_data)
Importance of components:
                          Comp.1     Comp.2     Comp.3      Comp.4
Standard deviation     2.0494032 0.49097143 0.27872586 0.153870700
Proportion of Variance 0.9246187 0.05306648 0.01710261 0.005212184
Cumulative Proportion  0.9246187 0.97768521 0.99478782 1.000000000
```

From the Proportion of Variance, we see that the first component has an importance of **92.5%** in predicting the class while the second principal component has an importance of **5.3%** and so on. This means that using just the first component instead of all the 4 features will make our model accuracy to be about **92.5%** while we use only one-fourth of the entire set of features.

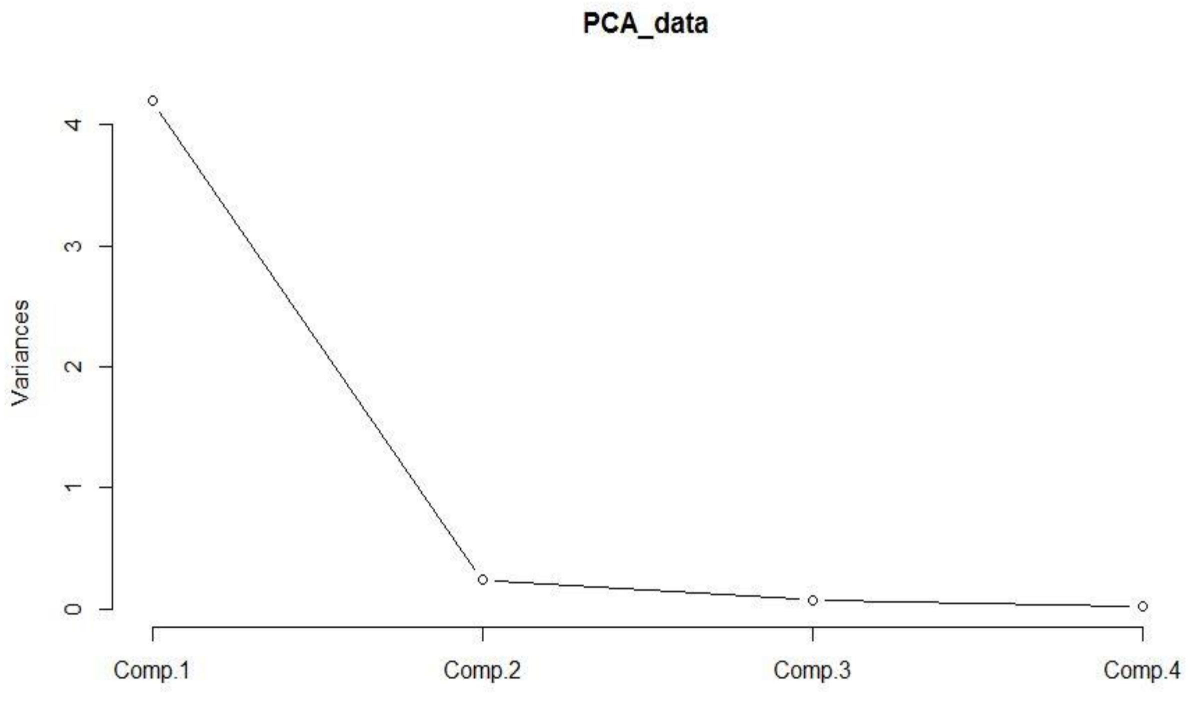
If we want the higher accuracy, we can take the first two components together and obtain a cumulative accuracy of up to **97.7%.** We can also understand how our features are transformed by using the biplot function on our model.

```
> biplot (PCA_data)
```



PCA feature transformation

```
> screeplot(PCA_data, type="lines")
```



principle components

This plot shows the bend at the second principal component.
Let us now fit two naive Bayes models.

1. one over the entire data.
2. The second on the first principal component.

We will calculate the difference in accuracy between these two models.

```
#Select the first principal component for the second model
> model2 = PCA_data$loadings[,1]

#For the second model, we need to calculate scores by multiplying our loadings with the data
> model2_scores <- as.matrix(data_iris) %*% model2

#Loading libraries for naiveBayes model
> library(class)
> install.packages("e1071")
> library(e1071)

#Fitting the first model over the entire data
> mod1<-naiveBayes(iris[,1:4], iris[,5])

#Fitting the second model using the first principal component
> mod2<-naiveBayes(model2_scores, iris[,5])

# Accuracy for the first model
>table(predict(mod1, iris[,1:4]), iris[,5])

             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          3        47

# Accuracy for the second model
>table(predict(mod2, model2_scores), iris[,5])

             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         46         5
  virginica       0          4        45
```

All Command:

```
data_iris <- iris[1:4]
Cov_data <- cov(data_iris )
# Find out the eigenvectors and eigenvalues using the covariance matrix
Eigen_data <- eigen(Cov_data)
# Using the inbuilt function
PCA_data <- princomp(data_iris ,cor="False")
# Let’s now compare the output variances
Eigen_data$values
PCA_data$sdev^2
PCA_data$loadings[,1:4]
Eigen_data$vectors
summary(PCA_data)
biplot (PCA_data)

screeplot(PCA_data, type="lines")
#Select the first principal component for the second model
model2 = PCA_data$loadings[,1]
#For the second model, we need to calculate scores by multiplying our loadings with the data
model2_scores <- as.matrix(data_iris) %*% model2

#Loading libraries for naiveBayes model
library(class)
install.packages("e1071")
library(e1071)

#Fitting the first model over the entire data
mod1<-naiveBayes(iris[,1:4], iris[,5])
#Fitting the second model using the first principal component
mod2<-naiveBayes(model2_scores, iris[,5])
# Accuracy for the first model
table(predict(mod1, iris[,1:4]), iris[,5])
# Accuracy for the second model
table(predict(mod2, model2_scores), iris[,5])
```