More_data_mining

Dr. Niladri Chakraborty

2025-05-15

1. Artificial Neural Network

How to choose the number of hidden layers and the number of neurons in each layer?

The choice of the number of hidden layers and the number of neurons in each hidden layer (the architecture of the neural network) is often more of an art than a science. It can depend on various factors, including the complexity of the problem, the size and dimensionality of the data, and computational constraints.

Here are some general guidelines:

Number of Hidden Layers: For many problems, a single hidden layer is sufficient. In fact, a neural network with one hidden layer can approximate any continuous function given enough neurons in the layer. However, additional hidden layers can make the network more efficient by reducing the number of required neurons and can help capture more complex features. Deep networks (with many layers) have been shown to perform exceptionally well on tasks such as image and speech recognition.

Number of Neurons in the Hidden Layers: As a very rough rule of thumb, the number of neurons in the hidden layers can be somewhere between the size of the input layer and the size of the output layer. Some common strategies include setting the number of neurons to be the mean of the input and output neurons, or the number of input neurons divided by 2. You may also start with a small number of neurons and gradually increase the number until the network performance no longer improves significantly.

In the end, the best network architecture often comes from experimenting and fine-tuning these parameters based on the performance of the model on validation data.

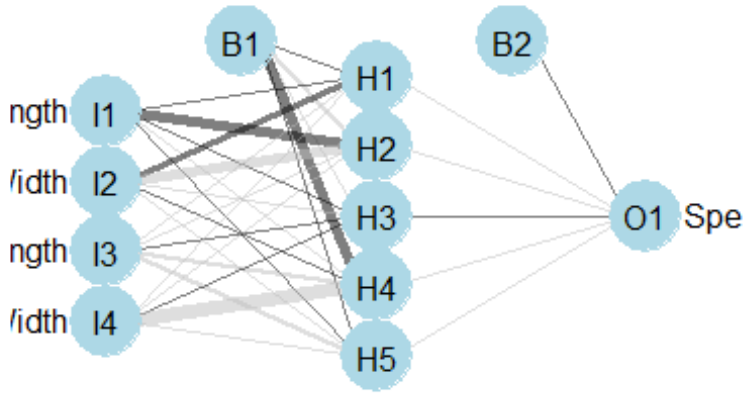
Predict the onset of diabetes:

Let's use the iris dataset which is built into R. This dataset contains measurements for 150 iris flowers from three different species.

Before training the neural network model, you'll want to check for and handle any missing values or NaN's in your dataset.

```
# Load necessary libraries
library(neuralnet)
```

```r
library(NeuralNetTools)

# Load the iris dataset
data(iris)

# Convert Species to numerical
iris$Species <- as.numeric(factor(iris$Species))

# Split the data into a training set (70% of the data) and a test set (30% of
the data)
set.seed(12345)
trainIndex <- sample(1:nrow(iris), nrow(iris)*0.7)
trainData <- iris[trainIndex, ]
testData <- iris[-trainIndex, ]

# Scale the training and test datasets
maxs <- apply(trainData, 2, max)
mins <- apply(trainData, 2, min)

scaled_trainData <- as.data.frame(scale(trainData, center = mins, scale =
maxs - mins))
scaled_testData <- as.data.frame(scale(testData, center = mins, scale = maxs
- mins)) # use the same scaling parameters

# Specify the formula
vars <- names(trainData)
f <- as.formula(paste("Species ~", paste(vars[!vars %in% "Species"], collapse
= " + ")))

# Train the neural network
set.seed(12345)
nn <- neuralnet(f, data = scaled_trainData, hidden = 5)

# Plot the neural network
plotnet(nn, alpha = 0.5)
```




```
# Generate predictions on the test set
testPredictors <- scaled_testData[, vars[!vars %in% "Species"]]
predictions <- predict(nn, testPredictors)

# Rescale the predicted values
predicted <- predictions * (maxs["Species"] - mins["Species"]) +
mins["Species"]

# Round the predicted values and clamp them to the range 1-3
predicted_rounded <- pmin(pmax(round(predicted), 1), 3)

# Calculate the accuracy
actuals <- testData$Species
accuracy <- mean(predicted_rounded == actuals)
print(accuracy)

## [1] 0.9777778
```

2. Support vector machine modeling with the ‘PimaIndiansDiabetes’ dataset.

Let us use the Pima Indians Diabetes dataset from the mlbench package instead.

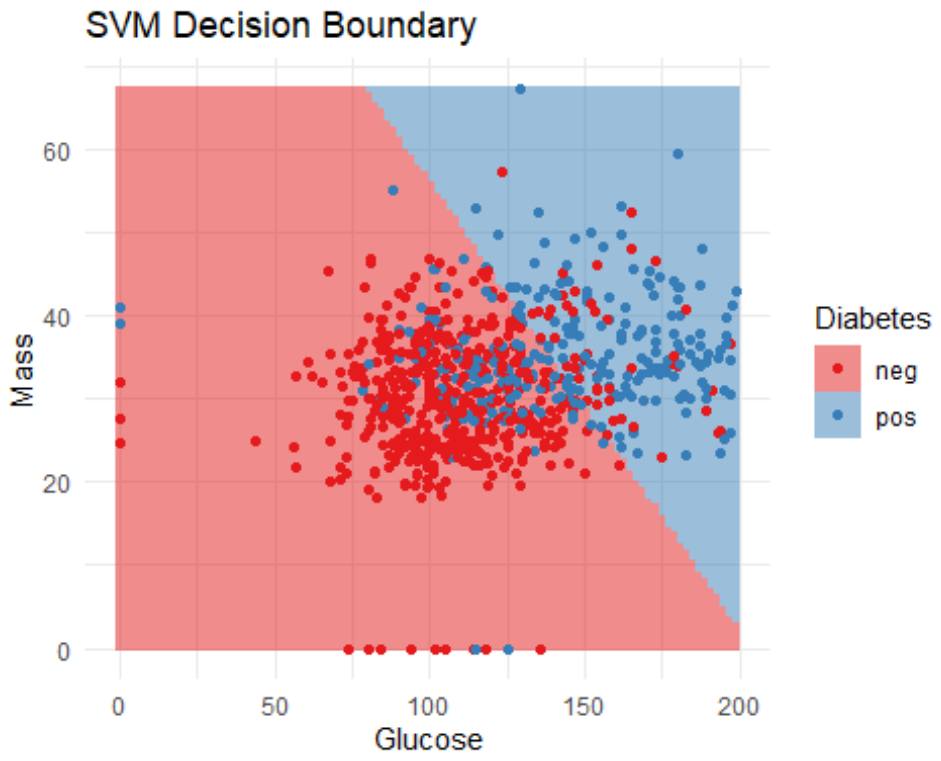
This dataset is a binary classification problem where all of the attributes are numeric and have different scales.

```
# Load necessary libraries
library(e1071) # For svm() function
library(caret) # For createDataPartition() function

## Warning: package 'caret' was built under R version 4.3.3

## Loading required package: ggplot2

## Loading required package: lattice

library(mlbench) # For PimaIndiansDiabetes dataset

# Load Pima Indians Diabetes dataset
data(PimaIndiansDiabetes)
diabetes <- PimaIndiansDiabetes

# Split the data into a training set (70% of the data) and a test set (30% of
the data)
set.seed(12345)
trainIndex <- createDataPartition(diabetes$diabetes, p = 0.7, list = FALSE)
trainData <- diabetes[trainIndex, ]
testData <- diabetes[-trainIndex, ]

# Train the SVM model
svm_model <- svm(diabetes ~ ., data = trainData, kernel = "radial")

# Generate predictions on the test set
predictions <- predict(svm_model, testData)

# Calculate the accuracy
accuracy <- sum(predictions == testData$diabetes) / nrow(testData)
print(accuracy)

## [1] 0.773913
```

Next, we will create a SVM plot with the ggplot() function.

```
# Load necessary libraries
library(e1071) # For svm() function
library(ggplot2) # For visualization
library(mlbench) # For PimaIndiansDiabetes dataset

# Load Pima Indians Diabetes dataset
data(PimaIndiansDiabetes)
diabetes <- PimaIndiansDiabetes
```

```r
# Select only two variables and the target
df <- diabetes[, c("glucose", "mass", "diabetes")]

# Train the SVM model
svm_model <- svm(diabetes ~ ., data = df, kernel = "linear")

# Create a grid for the plot
grid <- expand.grid(glucose = seq(min(df$glucose), max(df$glucose),
length.out = 100),
                    mass = seq(min(df$mass), max(df$mass), length.out = 100))

# Predict the grid to get decision values
grid$diabetes <- predict(svm_model, grid)

# Plot the decision boundary and the data points
ggplot(df, aes(glucose, mass)) +
  geom_tile(data = grid, aes(fill = diabetes), alpha = 0.5) +
  geom_point(aes(color = diabetes)) +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  theme_minimal() +
  labs(title = "SVM Decision Boundary", x = "Glucose", y = "Mass", fill =
"Diabetes", color = "Diabetes")
```



The plot generated by the previous code is a two-dimensional representation of the SVM decision boundary for the PimaIndiansDiabetes dataset. It uses only two variables from the dataset, glucose and mass.

The x-axis represents the glucose values and the y-axis represents the mass values. Each point in the plot corresponds to an observation in the dataset, and its position is determined by its glucose and mass values.

The colors of the points represent the actual class labels from the diabetes variable: one color for pos (diabetes positive) and another color for neg (diabetes negative).

The colored background represents the decision boundary of the SVM model. Each point in the background is colored according to the predicted class label for that combination of glucose and mass values. The boundary between the two colors is the decision boundary, where the model switches from predicting one class to predicting the other.

If the model is well-trained, we would expect the points to mostly match the color of the background, indicating that the model's predictions align with the actual class labels. Points that do not match the color of the background are instances where the model's prediction is incorrect.

Please note that this is a simplification because it only uses two variables from the dataset. In practice, SVMs often use many more variables, and the decision boundary is a hyperplane in a high-dimensional space that cannot be easily visualized.

3. Linear discriminant analysis with the IRIS dataset

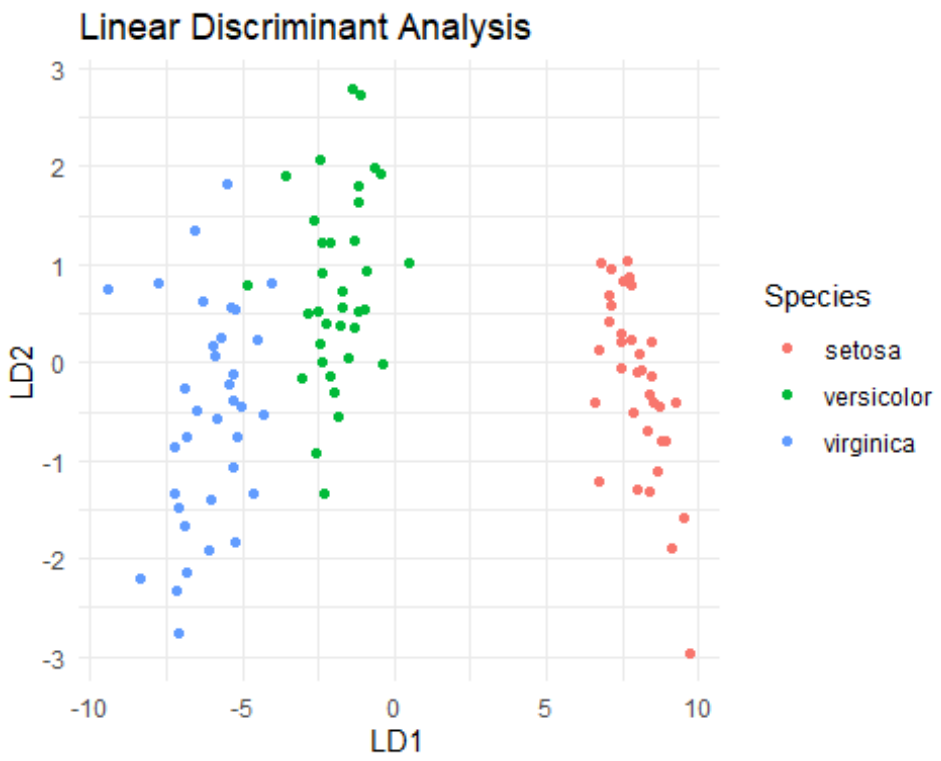
let's use the iris dataset for this example, which is built into R. This dataset contains measurements for 150 iris flowers from three different species. The four features measured are the lengths and the widths of the sepals and petals.

```
# Load necessary libraries
library(MASS) # For lda() function
library(caret) # For createDataPartition() function
library(ggplot2) # For visualization

# Load the iris dataset
data(iris)

# Split the data into a training set (70% of the data) and a test set (30% of
the data)
set.seed(12345)
trainIndex <- createDataPartition(iris$Species, p = 0.7, list = FALSE)
trainData <- iris[trainIndex, ]
testData <- iris[-trainIndex, ]
```

```r
# Train the LDA model
lda_model <- lda(Species ~ ., data = trainData)

# Generate predictions on the test set
predictions <- predict(lda_model, testData)$class

# Calculate the accuracy
accuracy <- sum(predictions == testData$Species) / nrow(testData)
print(accuracy)
```

```
## [1] 0.9777778
```

```r
# Create a data frame for plotting
plotData <- cbind(as.data.frame(predict(lda_model)$x), Species =
trainData$Species)

# Plot the LDA values
ggplot(plotData, aes(LD1, LD2)) +
  geom_point(aes(color = Species)) +
  theme_minimal() +
  labs(title = "Linear Discriminant Analysis", x = "LD1", y = "LD2", color =
"Species")
```



We're training a LDA model to predict the species of an iris flower based on the other variables in the iris dataset. The script includes steps to load the data, split it into a training

set and a test set, train the LDA model, generate predictions on the test set, calculate the accuracy of the predictions, and plot the LDA values.

4. Quadratic discriminant analysis

Quadratic Discriminant Analysis (QDA) is a variant of Linear Discriminant Analysis (LDA) where each class uses its own estimate of variance (or covariance when you have more than one input variable). This leads to quadratic decision boundaries, hence the name Quadratic Discriminant Analysis.

Let's use the PimaIndiansDiabetes dataset from the mlbench package for this example, which contains records for 768 Pima Indian women and whether they tested positive or negative for diabetes.

```
# Load necessary libraries
library(MASS) # For qda() function
library(mlbench) # For PimaIndiansDiabetes dataset
library(caret) # For createDataPartition() function
library(ggplot2) # For visualization

# Load the Pima Indians Diabetes dataset
data(PimaIndiansDiabetes)
diabetes <- PimaIndiansDiabetes

# Split the data into a training set (70% of the data) and a test set (30% of
the data)
set.seed(12345)
trainIndex <- createDataPartition(diabetes$diabetes, p = 0.7, list = FALSE)
trainData <- diabetes[trainIndex, ]
testData <- diabetes[-trainIndex, ]

# Train the QDA model
qda_model <- qda(diabetes ~ ., data = trainData)

# Generate predictions on the test set
predictions <- predict(qda_model, testData)$class

# Calculate the accuracy
accuracy <- sum(predictions == testData$diabetes) / nrow(testData)
print(accuracy)
```

```
## [1] 0.7565217
```

In QDA, unlike LDA, there is no direct equivalent to the linear discriminants that can be plotted. The reason is that QDA does not combine the variables linearly.

5. Principal component analysis

Principal Component Analysis (PCA) is a popular method for dimensionality reduction, i.e., to reduce the number of variables in your dataset. It works by creating new uncorrelated variables that successively maximize variance.

Let's use the USArrests dataset for this example, which is built into R and contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

```r
# Load the USArrests dataset
data(USArrests)

# Normalize the data
USArrests_scaled <- scale(USArrests)

# Perform PCA
pca_model <- prcomp(USArrests_scaled, center = TRUE, scale. = TRUE)

## The PCA scores, i.e., the coordinates of each city/state in the new PCA space
pca_model$x
```

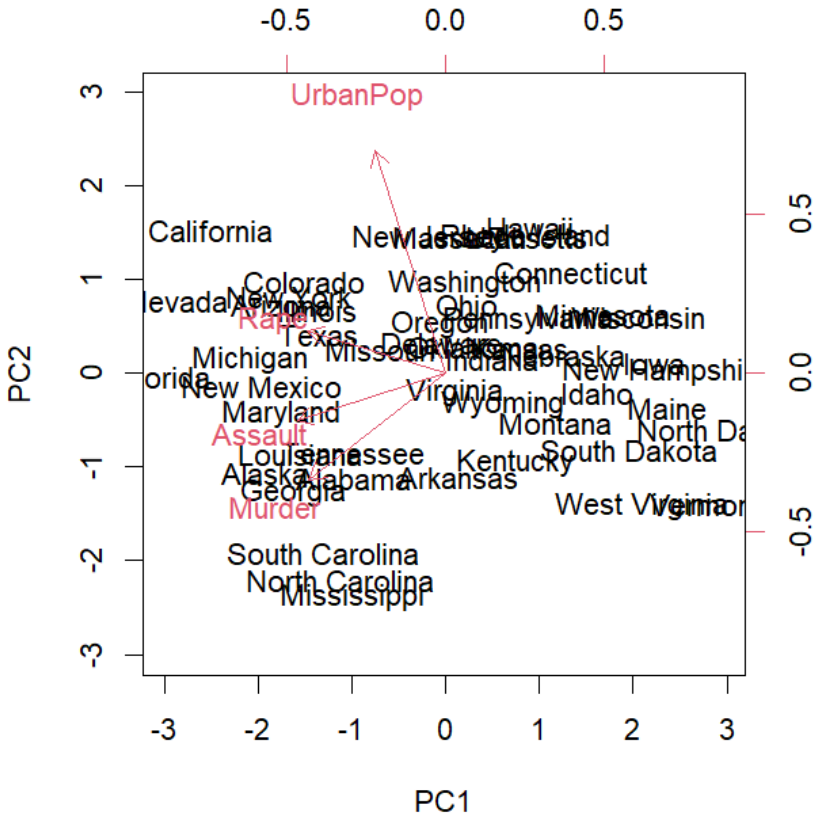
```
##                        PC1         PC2         PC3          PC4
## Alabama        -0.97566045 -1.12200121  0.43980366  0.154696581
## Alaska         -1.93053788 -1.06242692 -2.01950027 -0.434175454
## Arizona        -1.74544285  0.73845954 -0.05423025 -0.826264240
## Arkansas        0.13999894 -1.10854226 -0.11342217 -0.180973554
## California     -2.49861285  1.52742672 -0.59254100 -0.338559240
## Colorado       -1.49934074  0.97762966 -1.08400162  0.001450164
## Connecticut     1.34499236  1.07798362  0.63679250 -0.117278736
## Delaware       -0.04722981  0.32208890  0.71141032 -0.873113315
## Florida        -2.98275967 -0.03883425  0.57103206 -0.095317042
## Georgia        -1.62280742 -1.26608838  0.33901818  1.065974459
## Hawaii          0.90348448  1.55467609 -0.05027151  0.893733198
## Idaho           1.62331903 -0.20885253 -0.25719021 -0.494087852
## Illinois       -1.36505197  0.67498834  0.67068647 -0.120794916
## Indiana         0.50038122  0.15003926 -0.22576277  0.420397595
## Iowa            2.23099579  0.10300828 -0.16291036  0.017379470
## Kansas          0.78887206  0.26744941 -0.02529648  0.204421034
## Kentucky        0.74331256 -0.94880748  0.02808429  0.663817237
## Louisiana      -1.54909076 -0.86230011  0.77560598  0.450157791
## Maine           2.37274014 -0.37260865  0.06502225 -0.327138529
## Maryland       -1.74564663 -0.42335704  0.15566968 -0.553450589
## Massachusetts   0.48128007  1.45967706  0.60337172 -0.177793902
## Michigan       -2.08725025  0.15383500 -0.38100046  0.101343128
## Minnesota       1.67566951  0.62590670 -0.15153200  0.066640316
## Mississippi    -0.98647919 -2.36973712  0.73336290  0.213342049
## Missouri       -0.68978426  0.26070794 -0.37365033  0.223554811
```

```
## Montana         1.17353751 -0.53147851 -0.24440796  0.122498555
## Nebraska        1.25291625  0.19200440 -0.17380930  0.015733156
## Nevada         -2.84550542  0.76780502 -1.15168793  0.311354436
## New Hampshire   2.35995585  0.01790055 -0.03648498 -0.032804291
## New Jersey     -0.17974128  1.43493745  0.75677041  0.240936580
## New Mexico     -1.96012351 -0.14141308 -0.18184598 -0.336121113
## New York       -1.66566662  0.81491072  0.63661186 -0.013348844
## North Carolina -1.11208808 -2.20561081  0.85489245 -0.944789648
## North Dakota    2.96215223 -0.59309738 -0.29824930 -0.251434626
## Ohio            0.22369436  0.73477837  0.03082616  0.469152817
## Oklahoma        0.30864928  0.28496113  0.01515592  0.010228476
## Oregon         -0.05852787  0.53596999 -0.93038718 -0.235390872
## Pennsylvania    0.87948680  0.56536050  0.39660218  0.355452378
## Rhode Island    0.85509072  1.47698328  1.35617705 -0.607402746
## South Carolina -1.30744986 -1.91397297  0.29751723 -0.130145378
## South Dakota    1.96779669 -0.81506822 -0.38538073 -0.108470512
## Tennessee      -0.98969377 -0.85160534 -0.18619262  0.646302674
## Texas          -1.34151838  0.40833518  0.48712332  0.636731051
## Utah            0.54503180  1.45671524 -0.29077592 -0.081486749
## Vermont         2.77325613 -1.38819435 -0.83280797 -0.143433697
## Virginia        0.09536670 -0.19772785 -0.01159482  0.209246429
## Washington      0.21472339  0.96037394 -0.61859067 -0.218628161
## West Virginia   2.08739306 -1.41052627 -0.10372163  0.130583080
## Wisconsin       2.05881199  0.60512507  0.13746933  0.182253407
## Wyoming         0.62310061 -0.31778662  0.23824049 -0.164976866
```

```
# Print a summary of the PCA model
summary(pca_model)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.5749 0.9949 0.59713 0.41645
## Proportion of Variance 0.6201 0.2474 0.08914 0.04336
## Cumulative Proportion  0.6201 0.8675 0.95664 1.00000
```

```
# Create a biplot of the PCA model
biplot(pca_model, scale = 0)
```



6. Cluster analysis with the USarrests dataset.

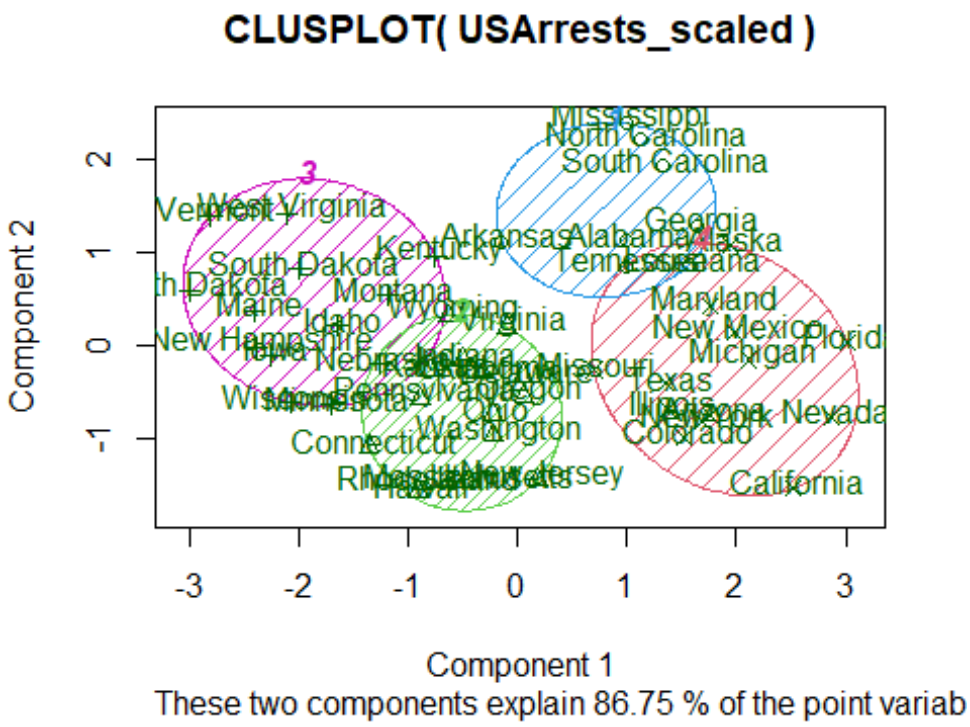
let's use the USArrests dataset for this example. This dataset is built into R and contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

```r
# Load necessary libraries
library(cluster) # For clusplot() function

# Load the USArrests dataset
data(USArrests)

# Normalize the data
USArrests_scaled <- scale(USArrests)

# Perform K-means clustering with 4 clusters
set.seed(12345) # For reproducibility
kmeans_model <- kmeans(USArrests_scaled, centers = 4)
```

```r
# Print the cluster assignments
print(kmeans_model$cluster)
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              4              4              1              4
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              4              2              2              4              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              3              4              2              3
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              3              1              3              4
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              4              3              1              4
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              3              3              4              3              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              4              4              1              3              2
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              2              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              3              1              4              2              3
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              3              3              2
```

```r
# Plot the clusters
clusplot(USArrests_scaled, kmeans_model$cluster, color=TRUE, shade=TRUE,
labels=2, lines=0)
```



we're performing K-means clustering on the USArrests dataset. The script includes steps to load and normalize the data, perform K-means clustering with 4 clusters, print the cluster assignments, and plot the clusters.

The plot produced by the code above is a 2D representation of the clusters formed by the K-means algorithm on the scaled USArrests dataset. This plot is generated using the first two principal components, which are linear combinations of the original variables that capture the maximum amount of variance in the data.

Each point in the plot represents a US state, and the points are colored according to their cluster assignments. The clusters represent groups of states with similar patterns of arrests for assault, murder, and rape, and similar percentages of the population living in urban areas. **States that are closer to each other in this plot have more similar crime statistics and urban population percentages**.

Note that the number of clusters (4 in this case) is a parameter that needs to be chosen carefully. Different numbers of clusters can lead to different insights about the data, and there are various methods to choose the optimal number of clusters, such as the elbow method and the silhouette method.

7. Analysis of variance

```r
# Load the data
data(iris)

# Perform one-way ANOVA
anova_result <- aov(Sepal.Length ~ Species, data = iris)

# Print a summary of the ANOVA model
summary(anova_result)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Species       2  63.21  31.606   119.3 <2e-16 ***
## Residuals   147  38.96   0.265
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We're performing one-way ANOVA on the Sepal.Length variable, which represents the sepal length, with respect to the Species variable, which represents the iris species.

The aov() function fits the ANOVA model, and the summary() function prints a summary of the ANOVA model.

This information helps assess whether there are significant differences in the mean sepal length across the different iris species.

8. Multivariate analysis of variance

Multivariate analysis of variance (MANOVA) is an extension of the univariate analysis of variance (ANOVA). In a MANOVA, we have two or more dependent variables.

To illustrate MANOVA in R, we will use the built-in mtcars dataset. This data set consists of 32 observations on 11 variables for different models of cars.

```r
# Load the necessary library
library(stats) # For MANOVA() function
library(datasets) # For mtcars dataset

# Load the data
data(mtcars)

# Convert cyl and am variables to factors
mtcars$cyl <- factor(mtcars$cyl)
mtcars$am <- factor(mtcars$am)

# Fit the MANOVA model
manova_model <- manova(cbind(mpg, disp, hp) ~ cyl + am, data = mtcars)

# Print a summary of the MANOVA model
summary(manova_model)
```

```
##           Df  Pillai approx F num Df den Df    Pr(>F)
## cyl        2 1.11005   11.226      6     54 4.236e-08 ***
## am         1 0.45133    7.129      3     26  0.001196 **
## Residuals 28
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The associated p-values provide information about the significance of the effects of the independent variables on the dependent variables.