

R PROGRAMMING

SCS1621

Unit – IV

Machine Learning in R - Classification: Decision Trees, Random Forest, SVM – Clustering - Association Rule Mining - Outlier Detection.

Machine learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Some machine learning methods

Machine learning algorithms are often categorized as supervised or unsupervised.

Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

In contrast, **unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.

Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

R provides many built in functions to work with the well-known machine learning algorithms like Regression, Decision Tree, SVM, Clustering etc...

Classification

Classification is a data mining function that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. For example, a classification model could be used to identify loan applicants as low, medium, or high credit risks.

Following are the examples of cases where the data analysis task is Classification –

- A bank loan officer wants to analyze the data in order to know which customer (loan applicant) are risky or which are safe.
- A marketing manager at a company needs to analyze a customer with a given profile, who will buy a new computer.

In both of the above examples, a model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data and yes or no for marketing data.

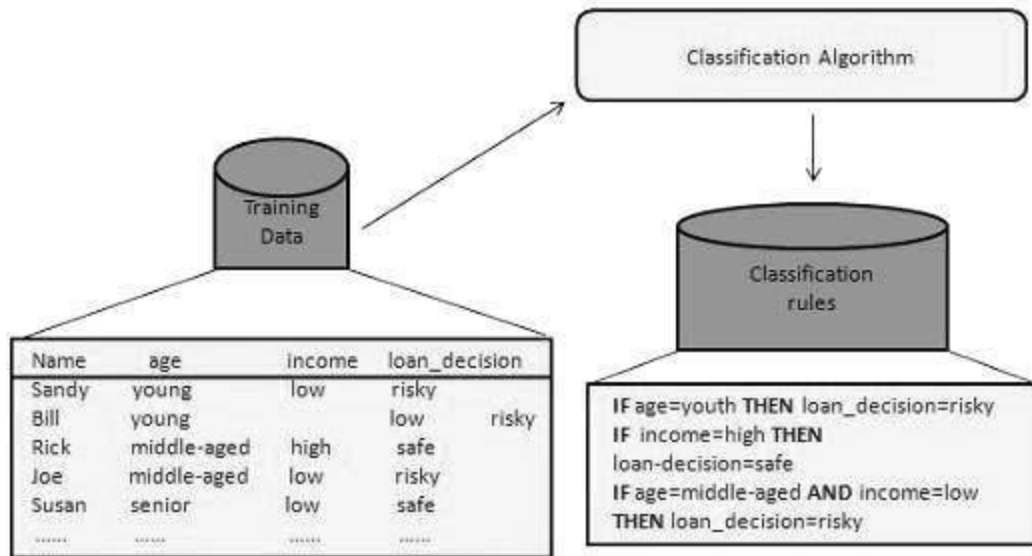
How Does Classification Works?

With the help of the bank loan application that we have discussed above, let us understand the working of classification. The Data Classification process includes two steps –

- Building the Classifier or Model
- Using Classifier for Classification

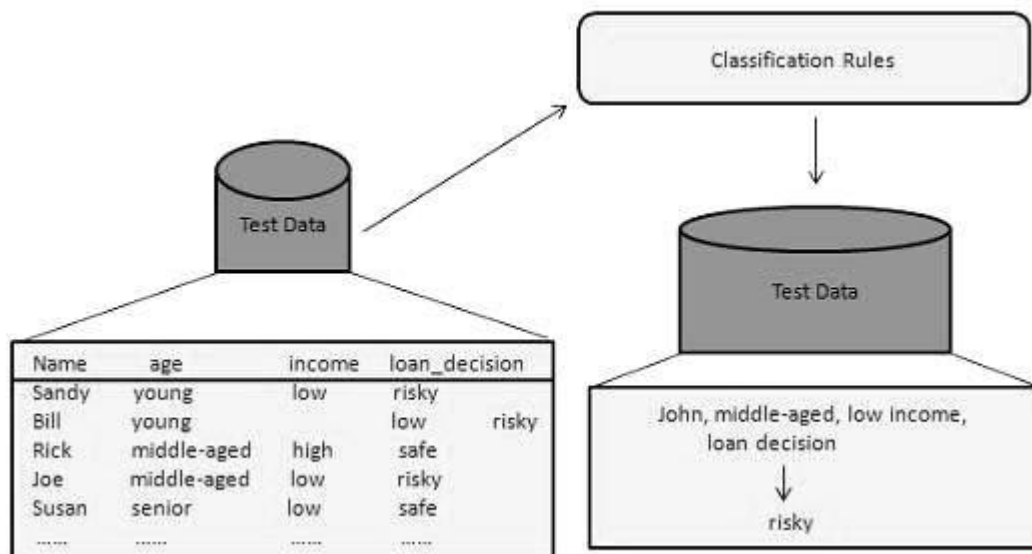
Building the Classifier or Model

- This step is the learning step or the learning phase.
- In this step the classification algorithms build the classifier.
- The classifier is built from the training set made up of database tuples and their associated class labels.
- Each tuple that constitutes the training set is referred to as a category or class. These tuples can also be referred to as sample, object or data points.



Using Classifier for Classification

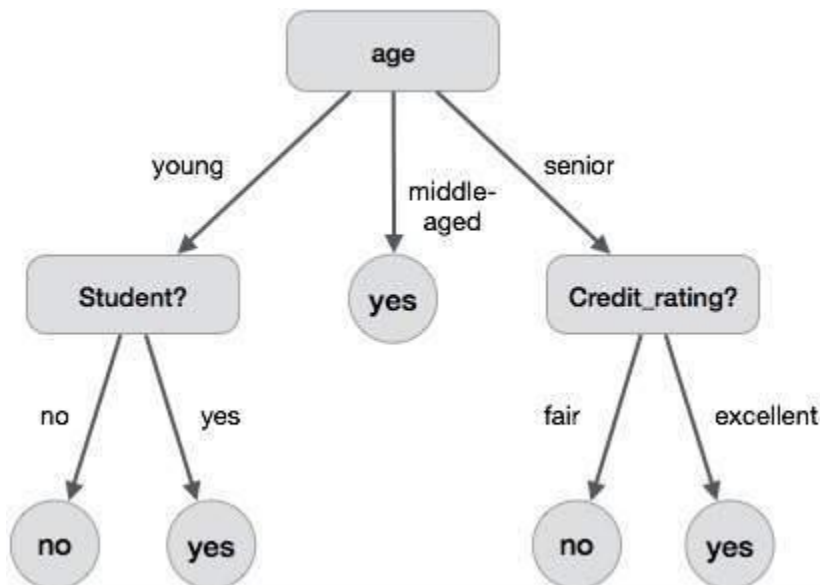
In this step, the classifier is used for classification. Here the test data is used to estimate the accuracy of classification rules. The classification rules can be applied to the new data tuples if the accuracy is considered acceptable.



Decision Tree

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

The following decision tree is for the concept `buy_computer` that indicates whether a customer at a company is likely to buy a computer or not. Each internal node represents a test on an attribute. Each leaf node represents a class.



The benefits of having a decision tree are as follows –

- It does not require any domain knowledge.
- It is easy to comprehend.
- The learning and classification steps of a decision tree are simple and fast.

Implementation of Decision Tree in R

The R package "**party**" is used to create decision trees.

Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

```
install.packages("party")
```

The package "party" has the function **ctree()** which is used to create and analyze decision tree.

Syntax

The basic syntax for creating a decision tree in R is –

```
ctree(formula, data)
```

Following is the description of the parameters used –

- **formula** is a formula describing the predictor and response variables.
- **data** is the name of the data set used.

Input Data

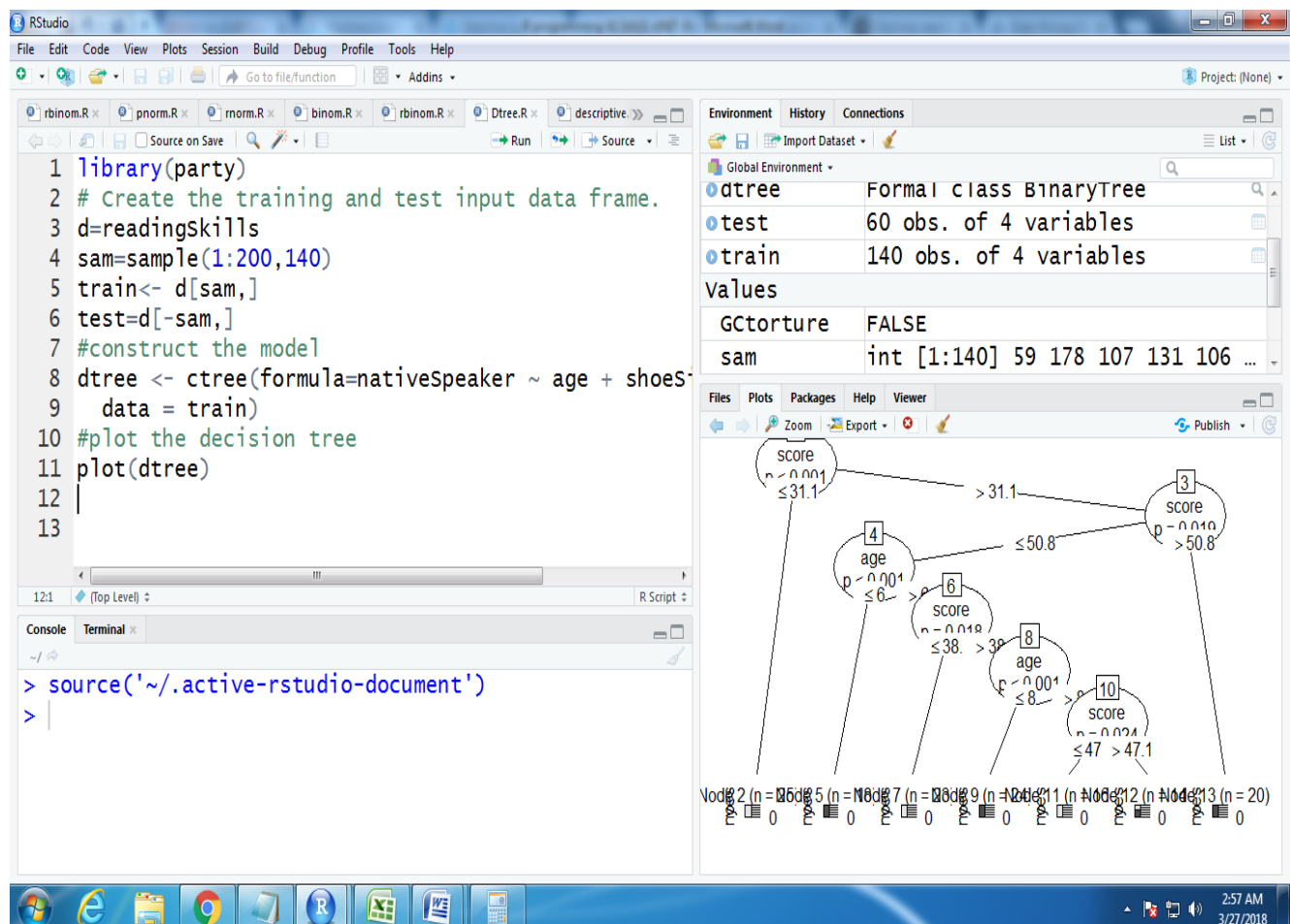
We will use the R in-built data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker or not.

```
> print(head(readingSkills))
  nativeSpeaker age shoesize  score
1          yes   5 24.83189 32.29385
2          yes   6 25.95238 36.63105
3           no  11 30.42170 49.60593
4          yes   7 28.66450 40.28456
5          yes  11 31.88207 55.46085
6          yes  10 30.07843 52.83124
```

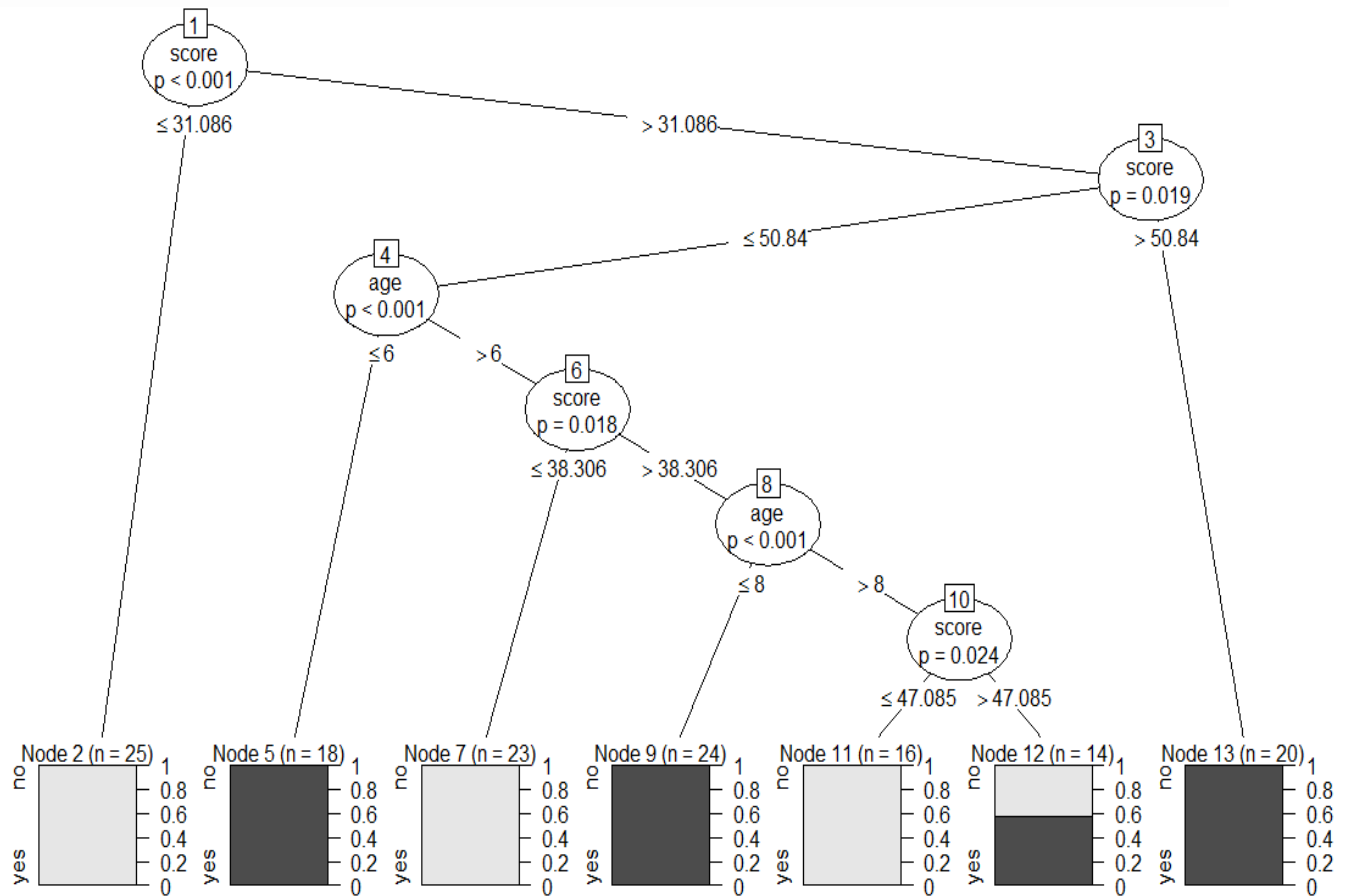
```
> dim(readingSkills)
```

```
[1] 200  4
```

Constructing Decision Tree Model in R



Output Decision Tree Structure:



Prediction Using Test data:

```
> pred=predict(dtree,test)
```

```
> pred
```

```
[1] yes yes yes yes no no no no yes no yes
```

```
[12] no no no no yes yes yes no no no no
```

```
[23] yes yes no no no no yes no yes yes no
```

```
[34] yes yes yes no yes yes no no yes no yes
```

```
[45] yes yes no yes yes yes no yes yes yes yes
```

```
[56] no yes yes no yes
```

Levels: no yes


```
> table(pred)
pred
no yes
27 33
```

Checking Accuracy of Constructed Model:

```
> acc=addmargins(table(pred,test$nativeSpeaker))
> acc
```

```
pred no yes Sum
no 27 0 27
yes 3 30 33

Sum 30 30 60
```

Accuracy=Total number of correctly classified observation/Total Observation

```
> value=57/60
> value
[1] 0.95
```

Conclusion:

Constructed Decision Tree 95% accurate

Random Forest

In the random forest approach, a large number of decision trees are created. Every observation is fed into every decision tree. The most common outcome for each observation is used as the final output. A new observation is fed into all the trees and taking a majority vote for each classification model.

An error estimate is made for the cases which were not used while building the tree. That is called an **OOB (Out-of-bag)** error estimate which is mentioned as a percentage.

The R package "**randomForest**" is used to create random forests.

Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

```
install.packages("randomForest")
```

The package "randomForest" has the function **randomForest()** which is used to create and analyze random forests.

Syntax

The basic syntax for creating a random forest in R is –

```
randomForest(formula, data)
```

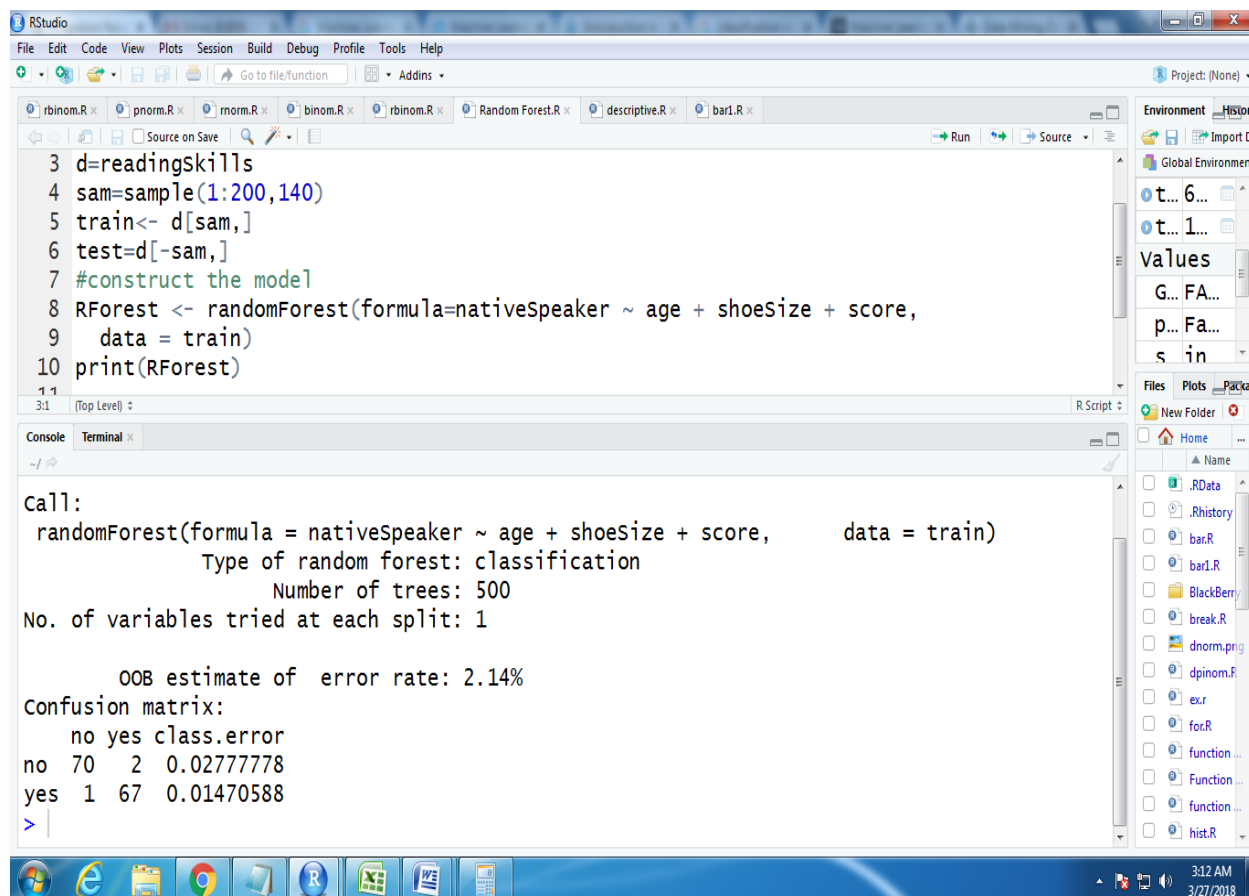
Following is the description of the parameters used –

- **formula** is a formula describing the predictor and response variables.
- **data** is the name of the data set used.
-

Input Data

We will use the R in-built data set named readingSkills to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker.

Construct Random Forest Model:



```
3 d=readingskills
4 sam=sample(1:200,140)
5 train<- d[sam,]
6 test=d[-sam,]
7 #construct the model
8 RForest <- randomForest(formula=nativeSpeaker ~ age + shoeSize + score,
9   data = train)
10 print(RForest)
```

Call:

```
randomForest(formula = nativeSpeaker ~ age + shoeSize + score,      data = train)
              Type of random forest: classification
              Number of trees: 500
No. of variables tried at each split: 1

OOB estimate of error rate: 2.14%
Confusion matrix:
      no yes class.error
no  70  2  0.02777778
yes  1  67  0.01470588
>
```

Prediction using Constructed Random Forest Model using Test Data:

```
> pred=predict(RForest,test)
```

```
> pred
```

```
1  2  4  5  8  9 10 23 27 30 35 39 42 43 44 46 47 48 57 60 61 62
```

```
yes yes yes yes yes yes no no yes yes yes yes no no no yes yes no yes no yes no 68
70 71 72 76 77 79 84 86 89 96 100 101 105 107 109 120 129 131 132 133 141
```

```
no yes yes no yes yes no no yes yes no yes yes no yes no no yes no no yes no
```

```
142 147 152 154 166 170 173 174 187 188 190 191 193 194 197 200
```

no yes yes yes no no yes no yes yes yes no no no yes no

Levels: no yes

```
> table(pred)
```

pred

no yes

27 33

Checking Accuracy of the constructed Model

```
> acc=addmargins(table(pred,test$nativeSpeaker))
```

```
> acc
```

pred	no	yes	Sum
------	----	-----	-----

no	27	0	27
----	----	---	----

yes	1	32	33
-----	---	----	----

Sum	28	32	60
-----	----	----	----

Calculate Accuracy Value from the above table:

```
> accvalue=(27+32)/60
```

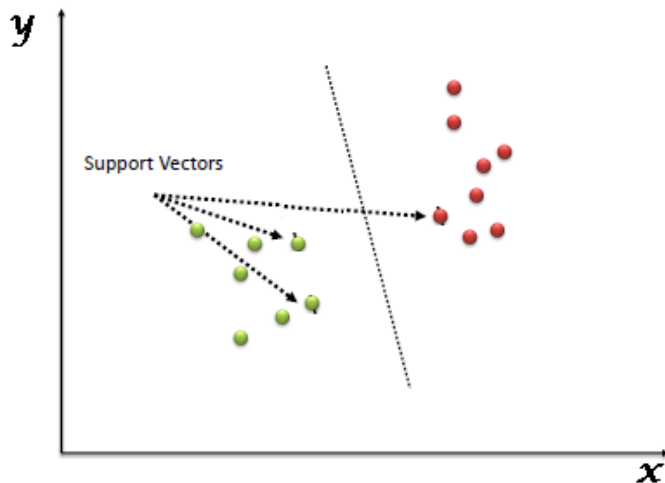
```
> accvalue
```

```
[1] 0.9833333
```

Conclusion: Hence the constructed model is 98% Accurate.

Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).



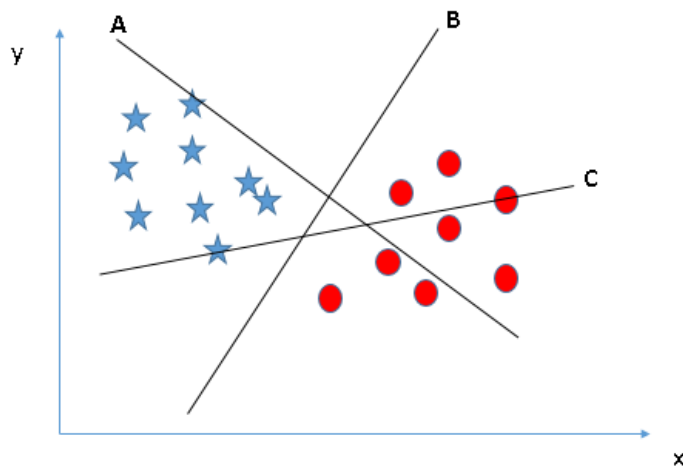
Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify

star

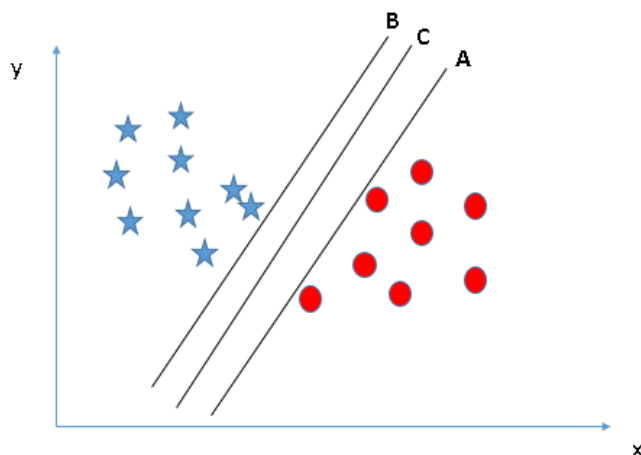
and

circle.

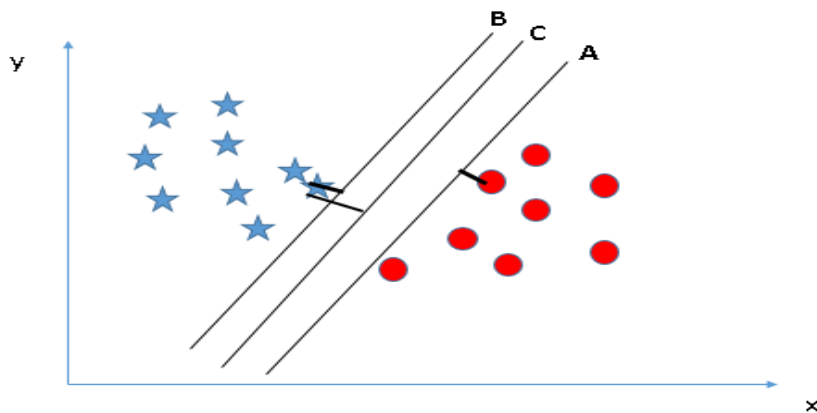


“Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.

Identify the right hyper-plane (Scenario-2): Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

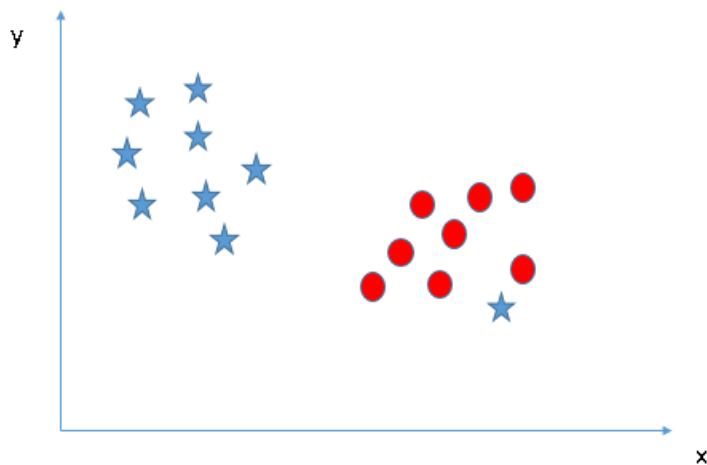


Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Consider the below snapshot:



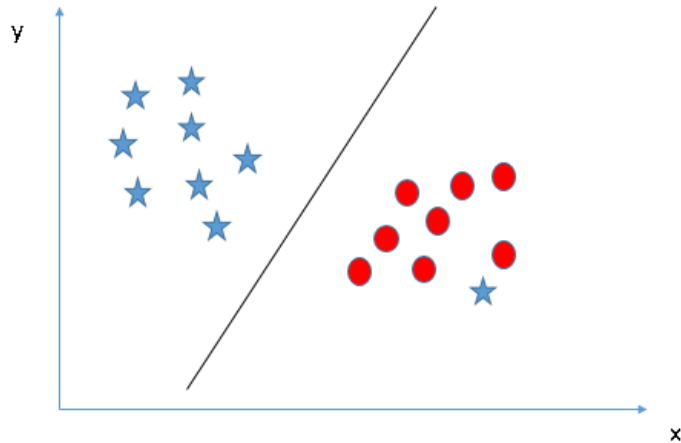
In the above figure, margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

Can we classify two classes (Scenario-3): In below figure , It is unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier.

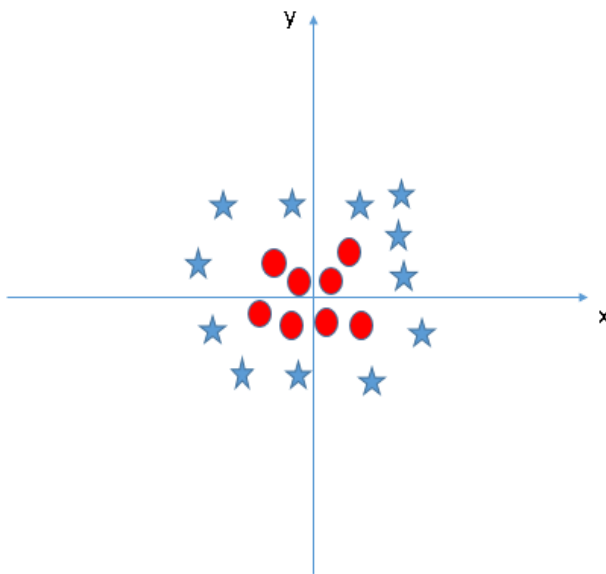


Here one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin.

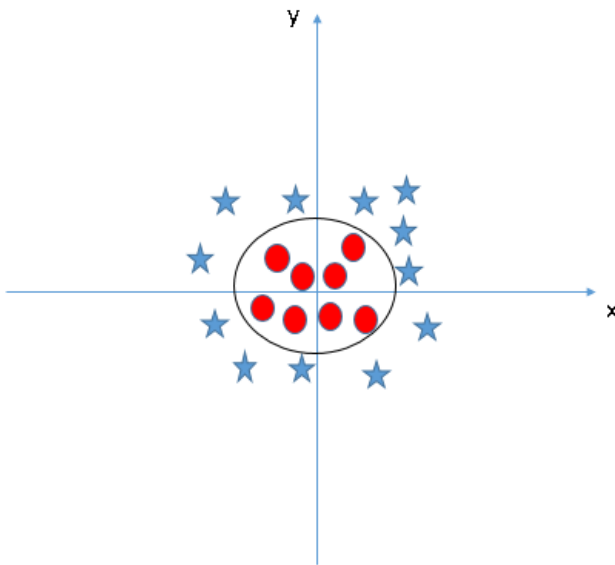
Hence, we can say, SVM is robust to outliers.



Find the hyper-plane to segregate to classes (Scenario-4): In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only \looked at the linear hyper-plane.



When we look at the hyper-plane in original input space it looks like a circle:



Support Vector Implementation in R

Install the required package e1071 for SVM implementation.

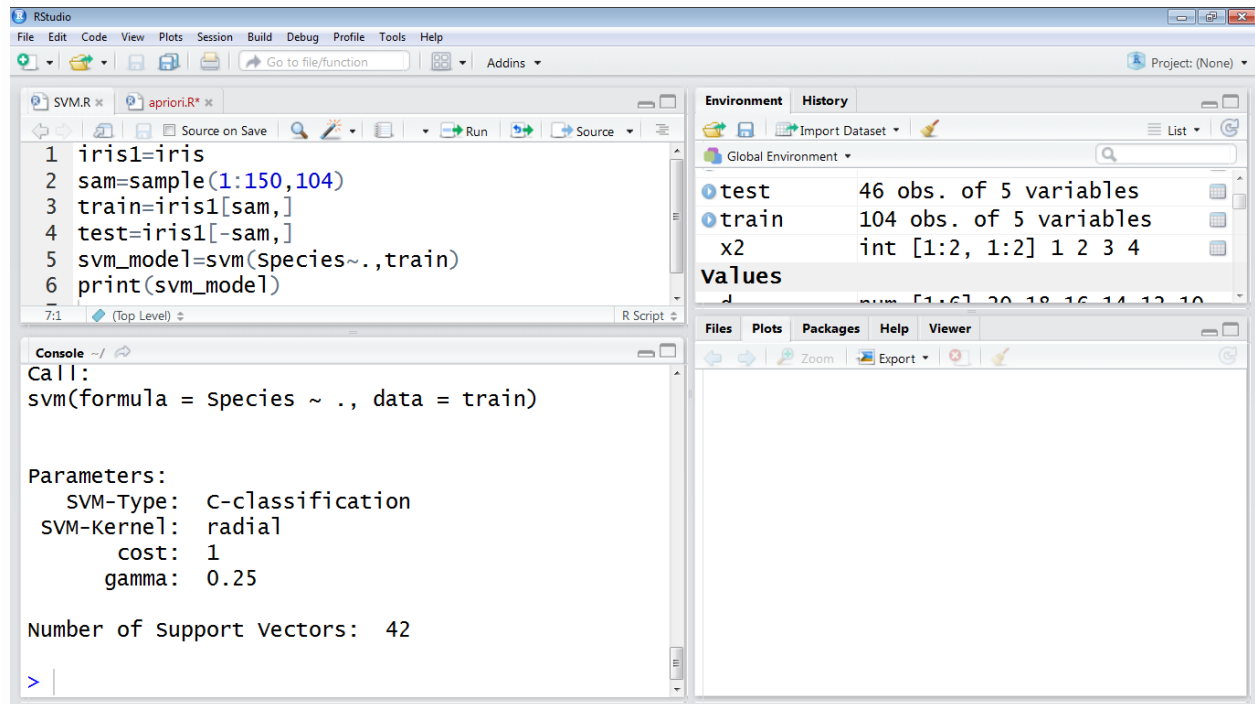
Syntax:

Install the package e1071.

```
>install.packages("e1071")
```

```
>library(e1071)
```

SVM Model Construction for iris dataset and it's output:



The screenshot shows the RStudio interface with the following components:

- Source Editor:** Contains the following R code:

```
1 iris1=iris
2 sam=sample(1:150,104)
3 train=iris1[sam,]
4 test=iris1[-sam,]
5 svm_model=svm(Species~.,train)
6 print(svm_model)
```
- Console:** Shows the execution of the SVM model construction:

```
Call:
svm(formula = Species ~ ., data = train)

Parameters:
  SVM-Type:  C-classification
 SVM-Kernel: radial
    cost:    1
   gamma:   0.25

Number of Support Vectors: 42
```
- Environment:** Lists the objects in the global environment:
 - `test`: 46 obs. of 5 variables
 - `train`: 104 obs. of 5 variables
 - `x2`: int [1:2, 1:2] 1 2 3 4

Predicting values for test data using constrected model:

```
> pred=predict(svm_model,test)
> table(pred)
pred
      setosa versicolor  virginica 
      15         19         12
```

Checking Accuracy of the constrected Model:

```
> addmargins(table(pred,test$Species))

pred      setosa versicolor virginica Sum
setosa      15         0         0    15
versicolor   0        16         3    19
virginica     0         0        12    12
Sum          15        16        15    46
```

Calculating Accuracy value from the above table:

```
> AccValue=(15+16+12)/46
> AccValue
[1] 0.9347826
```

Conclusion:

The Constructed model is 93% Accurate.

Clustering

Cluster is a group of objects that belongs to the same class. In other words, similar objects are grouped in one cluster and dissimilar objects are grouped in another cluster.

What is Clustering?

Clustering is the process of making a group of abstract objects into classes of similar objects.

- A cluster of data objects can be treated as one group.
- While doing cluster analysis, we first partition the set of data into groups based on data similarity and then assign the labels to the groups.
- The main advantage of clustering over classification is that, it is adaptable to changes and helps single out useful features that distinguish different groups.

Applications of Cluster Analysis

- Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.
- Clustering can also help marketers discover distinct groups in their customer base. And they can characterize their customer groups based on the purchasing patterns.
- In the field of biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionalities and gain insight into structures inherent to populations.
- Clustering also helps in identification of areas of similar land use in an earth observation database. It also helps in the identification of groups of houses in a city according to house type, value, and geographic location.
- Clustering also helps in classifying documents on the web for information discovery.

- Clustering is also used in outlier detection applications such as detection of credit card fraud.
- As a data mining function, cluster analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.

Requirements of Clustering in Data Mining

The following points throw light on why clustering is required in data mining

- **Scalability** – We need highly scalable clustering algorithms to deal with large databases.
- **Ability to deal with different kinds of attributes** – Algorithms should be capable to be applied on any kind of data such as interval-based (numerical) data, categorical, and binary data.
- **Discovery of clusters with attribute shape** – The clustering algorithm should be capable of detecting clusters of arbitrary shape. They should not be bounded to only distance measures that tend to find spherical cluster of small sizes.
- **High dimensionality** – The clustering algorithm should not only be able to handle low-dimensional data but also the high dimensional space.
- **Ability to deal with noisy data** – Databases contain noisy, missing or erroneous data. Some algorithms are sensitive to such data and may lead to poor quality clusters.
- **Interpretability** – The clustering results should be interpretable, comprehensible, and usable.

Clustering Methods

Clustering methods can be classified into the following categories –

- Partitioning Method
- Hierarchical Method
- Density-based Method
- Grid-Based Method
- Model-Based Method
- Constraint-based Method

Partitioning Method

Suppose we are given a database of 'n' objects and the partitioning method constructs 'k' partition of data. Each partition will represent a cluster and $k \leq n$. It means that it will classify the data into k groups, which satisfy the following requirements –

- Each group contains at least one object.
- Each object must belong to exactly one group.

Hierarchical Methods

This method creates a hierarchical decomposition of the given set of data objects. We can classify hierarchical methods on the basis of how the hierarchical decomposition is formed. There are two approaches here –

- Agglomerative Approach
- Divisive Approach

Agglomerative Approach

This approach is also known as the bottom-up approach. In this, we start with each object forming a separate group. It keeps on merging the objects or groups that are close to one another. It keep on doing so until all of the groups are merged into one or until the termination condition holds.

Divisive Approach

This approach is also known as the top-down approach. In this, we start with all of the objects in the same cluster. In the continuous iteration, a cluster is split up into smaller clusters. It is down until each object in one cluster or the termination condition holds. This method is rigid, i.e., once a merging or splitting is done, it can never be undone.

Approaches to Improve Quality of Hierarchical Clustering

Here are the two approaches that are used to improve the quality of hierarchical clustering –

- Perform careful analysis of object linkages at each hierarchical partitioning.
- Integrate hierarchical agglomeration by first using a hierarchical agglomerative algorithm to group objects into micro-clusters, and then performing macro-clustering on the micro-clusters.

Density-based Method

This method is based on the notion of density. The basic idea is to continue growing the given cluster as long as the density in the neighborhood exceeds some threshold, i.e., for each data point within a given cluster, the radius of a given cluster has to contain at least a minimum number of points.

Grid-based Method

In this, the objects together form a grid. The object space is quantized into finite number of cells that form a grid structure.

Advantage

- The major advantage of this method is fast processing time.
- It is dependent only on the number of cells in each dimension in the quantized space.

Model-based methods

In this method, a model is hypothesized for each cluster to find the best fit of data for a given model. This method locates the clusters by clustering the density function. It reflects spatial distribution of the data points.

This method also provides a way to automatically determine the number of clusters based on standard statistics, taking outlier or noise into account. It therefore yields robust clustering methods.

Constraint-based Method

In this method, the clustering is performed by the incorporation of user or application-oriented constraints. A constraint refers to the user expectation or the properties of desired clustering results. Constraints provide us with an interactive way of communication with the clustering process. Constraints can be specified by the user or the application requirement.

K-Means Clustering

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the k Means algorithm; an unsupervised learning algorithm.

(It will help if you think of items as points in an n-dimensional space). The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.

The algorithm works as follows:

1. First we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set .

K Means Clustering Implementaion in R.

Example:Cluster the iris data set using K mean clustering algorithm.

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

SVM.R x Untitled1.R x apriori.R x

Source on Save Run Source

```

1 iris1=iris[,1:4]
2 clusmodel=kmeans(iris1,3)
3 print(clusmodel)
4

```

4:1 (Top Level) R Script

Environment History

Global Environment

Data

- iris1 150 obs. of 4 variables
- test 46 obs. of 5 variables
- train 104 obs. of 5 variables
- x2 int [1:2, 1:2] 1 2 3 4

Files Plots Packages Help Viewer

Zoom Export

Console

```

> source('~/.active-rstudio-document')
K-means clustering with 3 clusters of sizes 38, 50, 62

Cluster means:
  Sepal.Length Sepal.Width Petal.Length
1  6.850000    3.073684    5.742105
2  5.006000    3.428000    1.462000
3  5.901613    2.748387    4.393548
  Petal.Width
1  2.071053
2  0.246000
3  1.433871

Clustering vector:
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[24] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[47] 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[70] 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

```

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

SVM.R x Untitled1.R x apriori.R x

Source on Save Run Source

```

1 iris1=iris[,1:4]
2 clusmodel=kmeans(iris1,3)
3 print(clusmodel)
4

```

4:1 (Top Level) R Script

Environment History

Global Environment

Data

- iris1 150 obs. of 4 variables
- test 46 obs. of 5 variables
- train 104 obs. of 5 variables
- x2 int [1:2, 1:2] 1 2 3 4

Files Plots Packages Help Viewer

Zoom Export

Console

```

 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[24] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[47] 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[70] 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[93] 3 3 3 3 3 3 3 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[116] 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[139] 3 1 1 1 3 1 1 1 3 1 1 3

within cluster sum of squares by cluster:
[1] 23.87947 15.15100 39.82097
(between_SS / total_SS = 88.4 %)

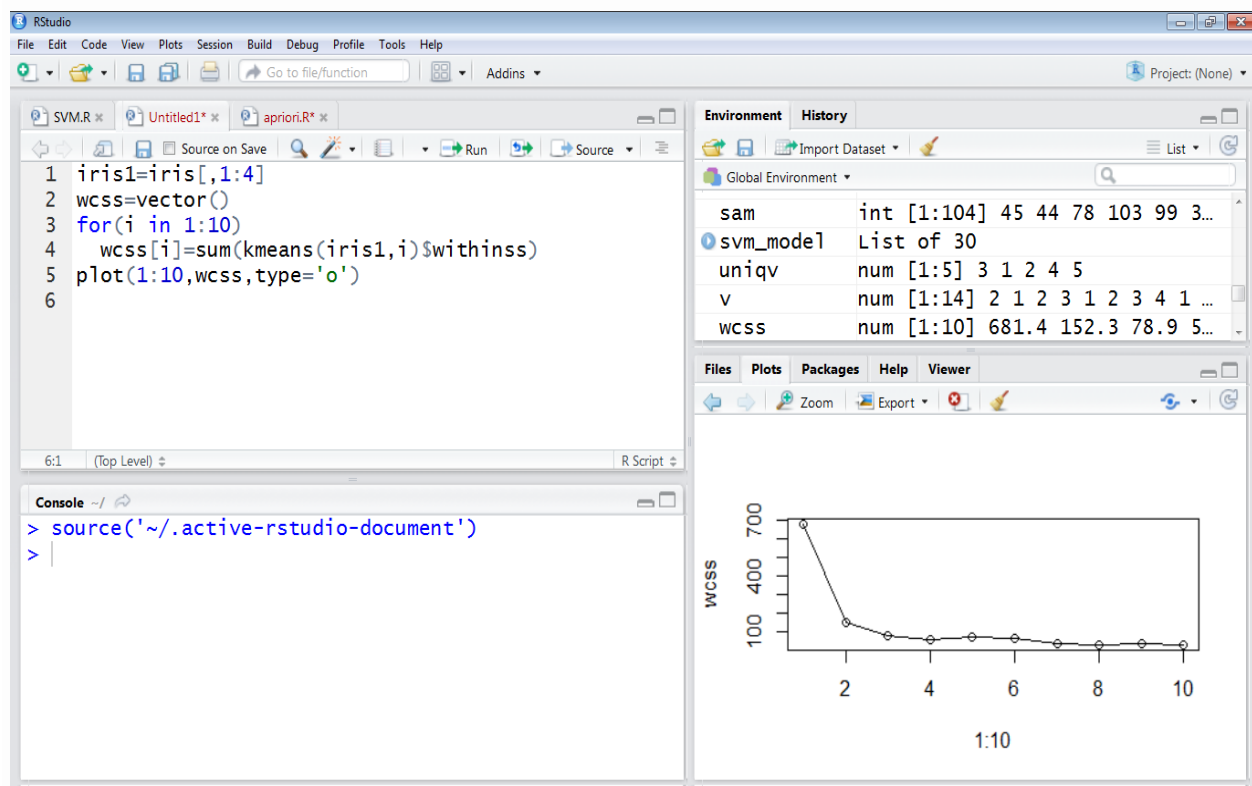
Available components:
[1] "cluster"      "centers"      "totss"
[4] "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
>

```


Find Optimal Number of clusters using Elbow Method:

One method to validate the number of clusters is the *elbow method*. The idea of the elbow method is to run k-means clustering on the dataset for a range of values of k (say, k from 1 to 10 in the examples above), and for each value of k calculate the sum of squared errors (SSE).

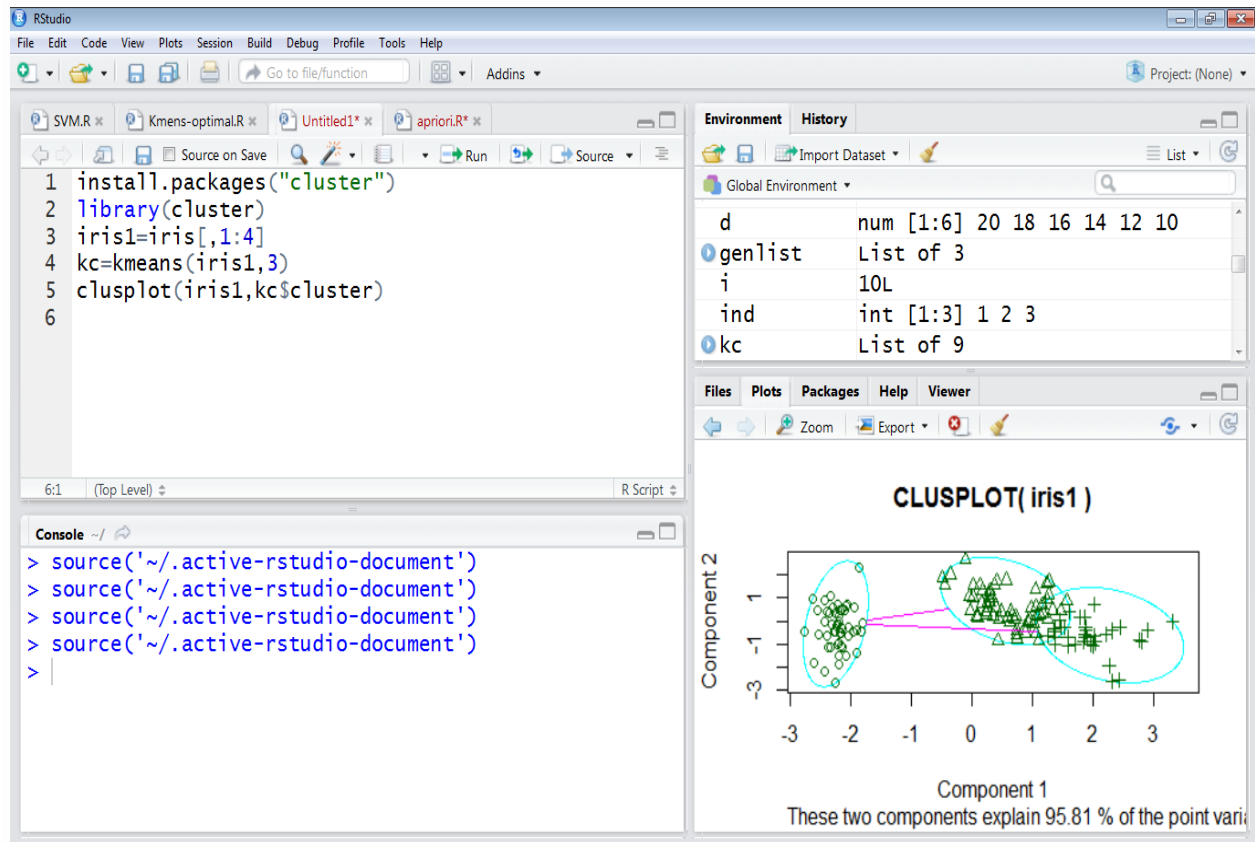
Then, plot a line chart of the SSE for each value of k . If the line chart looks like an arm, then the "elbow" on the arm is the value of k that is the best. The idea is that we want a small SSE, but that the SSE tends to decrease toward 0 as we increase k (the SSE is 0 when k is equal to the number of data points in the dataset, because then each data point is its own cluster, and there is no error between it and the center of its cluster). So our goal is to choose a small value of k that still has a low SSE, and the elbow usually represents where we start to have diminishing returns by increasing k .



Optimal number of clusters in the above example is 3.

Plotting cluster output:

Install the package cluster and use the function `clus plot()` to visualize clustering results.



Association Rule Mining

Association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. It is the way of analyzing and presenting strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, discover regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the

rule $\{\text{onions, potatoes}\} \Rightarrow \{\text{burger}\}$ found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, he or she is likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection and bioinformatics. As opposed to sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

Apriori Algorithm

The most famous algorithm for association rule learning is Apriori. It was proposed by Agrawal and Srikant in 1994. The input of the algorithm is a dataset of transactions where each transaction is a set of items. The output is a collection of association rules for which support and confidence are greater than some specified threshold. The name comes from the Latin phrase *a priori* (literally, "from what is before") because of one smart observation behind the algorithm: if the item set is infrequent, then we can be sure in advance that all its subsets are also infrequent.

You can implement Apriori with the following steps:

1. Count the support of all item sets of length 1, or calculate the frequency of every item in the dataset.
2. Drop the item sets that have support lower than the threshold.
3. Store all the remaining item sets.
4. Extend each stored item set by one element with all possible extensions. This step is known as candidate generation.

5. Calculate the support value of each candidate.
6. Drop all candidates below the threshold.
7. Drop all stored items from step 3 that have the same support as their extensions.
8. Add all the remaining candidates to storage.
9. Repeat steps 4 to step 8 until there are no more extensions with support greater than the threshold.

This is not a very efficient algorithm if you have a lot of data, but mobile applications are not recommended for use with big data anyway. This algorithm was influential in its time, and is also elegant and easy to understand today.

Implementation of Apriori Algorithm in R

import the package and use the package arules:

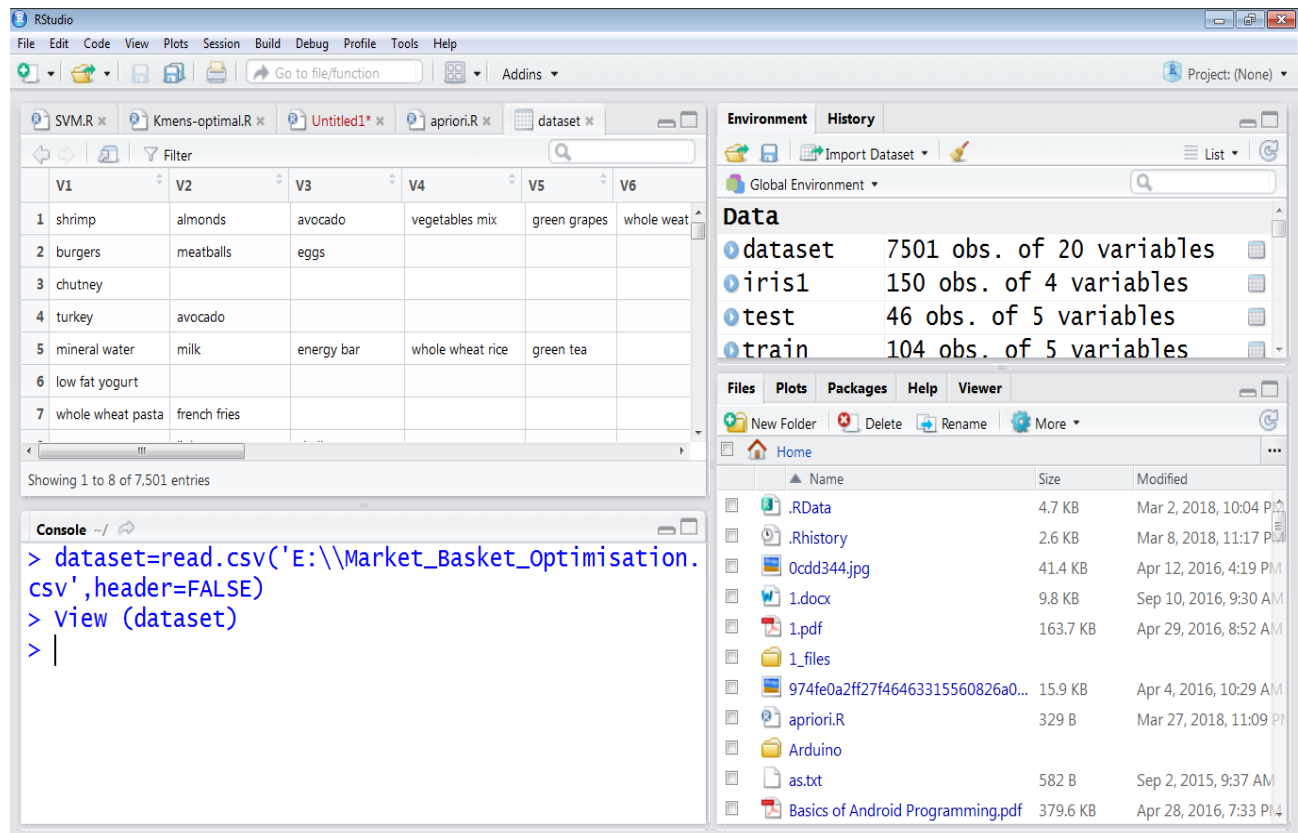
```
>install.packages("arules")
```

```
>library(arules)
```

Load the data set

Market _Basket_Optimisation data set should be downloaded from the below website.

www.superdatascience/machinelearning



Convert the dataset into sparse Matrix:

```
>
dataset=read.transactions('E:\\Market_Basket_Optimisation.csv',sep=",",rm.duplicates=TRUE)
distribution of transactions with duplicates:
1
5
> dataset
transactions in sparse format with
7501 transactions (rows) and
119 items (columns)
```

Get the Summary of the given data set:

```
> summary(dataset)
transactions as itemMatrix in sparse format with
7501 rows (elements/itemsets/transactions) and
119 columns (items) and a density of 0.03288973

most frequent items:
mineral water      eggs      spaghetti  french fries
      1788      1348      1306      1282
      chocolate    (Other)
      1229      22405
```

element (itemset/transaction) length distribution:

sizes	1	2	3	4	5	6	7	8	9	10	11	12	13
	1754	1358	1044	816	667	493	391	324	259	139	102	67	40
sizes	14	15	16	18	19	20							
	22	17	4	1	2	1							

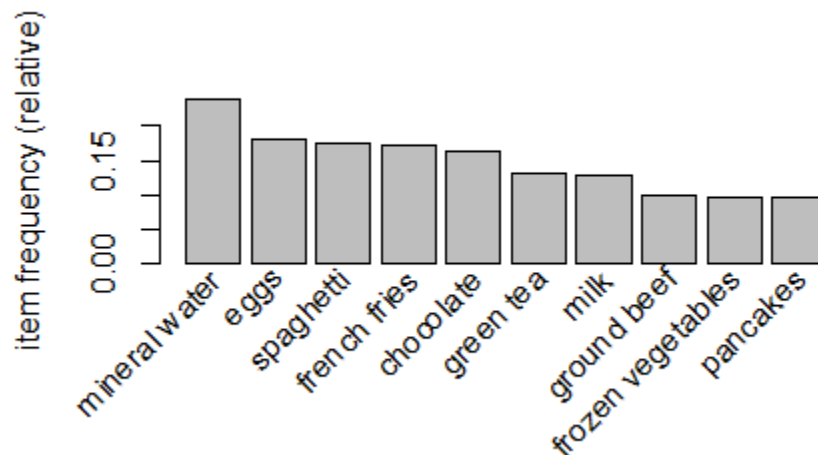
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	2.000	3.000	3.914	5.000	20.000

includes extended item information - examples:

	labels
1	almonds
2	antioxydant juice
3	asparagus

Plot ten items with Highest frequency :

> itemFrequencyPlot(dataset,topN=10)



Generate Association Rules with support =0.003 and confidence=0.8

> rules=apriori(data=dataset,parameter=list(support=0.003,confidence=0.8))

Parameter specification:

confidence	minval	smax	arem	aval	originalsupport	maxtime	support
0.8	0.1	1	none	FALSE	TRUE	5	0.003
minlen	maxlen	target	ext				
1	10	rules	FALSE				

Algorithmic control:

filter	tree	heap	memopt	load	sort	verbose
0.1	TRUE	TRUE	FALSE	TRUE	2	TRUE

Absolute minimum support count: 22

```
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
sorting and recoding items ... [115 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.00s].
writing ... [0 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

Generate Association Rules with support =0.003 and confidence=0.4

```
>
rules=apriori(data=dataset,parameter=list(support=0.003,confiden
ce=0.4))
Apriori
```

Parameter specification:

confidence	minval	smax	arem	aval	original	support	maxtime	support
0.4	0.1	1	none	FALSE		TRUE	5	0.003
minlen	maxlen	target	ext					
1	10	rules	FALSE					

Algorithmic control:

filter	tree	heap	memopt	load	sort	verbose
0.1	TRUE	TRUE	FALSE	TRUE	2	TRUE

Absolute minimum support count: 22

```
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
sorting and recoding items ... [115 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.00s].
writing ... [281 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

Print the top 10 rules

```
> inspect(sort(rules,by = 'lift')[1:10])
```

	lhs	rhs	support	confidence	lift	count
[1]	{mineral water,whole wheat pasta}	=> {olive oil}	0.003866151	0.4027778	6.115863	29
[2]	{spaghetti,tomato sauce}	=> {ground beef}	0.003066258	0.4893617	4.980600	23
[3]	{french fries,herb & pepper}	=> {ground beef}	0.003199573	0.4615385	4.697422	24
[4]	{cereals,spaghetti}	=> {ground beef}	0.003066258	0.4600000	4.681764	23
[5]	{frozen vegetables,mineral water,soup}	=> {milk}	0.003066258	0.6052632	4.670863	23
[6]	{chocolate,herb & pepper}	=> {ground beef}	0.003999467	0.4411765	4.490183	30
[7]	{chocolate,mineral water,shrimp}	=> {frozen vegetables}	0.003199573	0.4210526	4.417225	24
[8]	{frozen vegetables,mineral water,olive oil}	=> {milk}	0.003332889	0.5102041	3.937285	25
[9]	{cereals,ground beef}	=> {spaghetti}	0.003066258	0.6764706	3.885303	23
[10]	{frozen vegetables,soup}	=> {milk}	0.003999467	0.5000000	3.858539	30

Outlier Analysis

Outlier:

An outlier is a data object that deviates significantly from the rest of the objects, as if it were generated by a different mechanism. Data objects that are not outliers as “normal” or expected data. Similarly, we may refer to outliers as “abnormal” data.

Outlier Analysis:

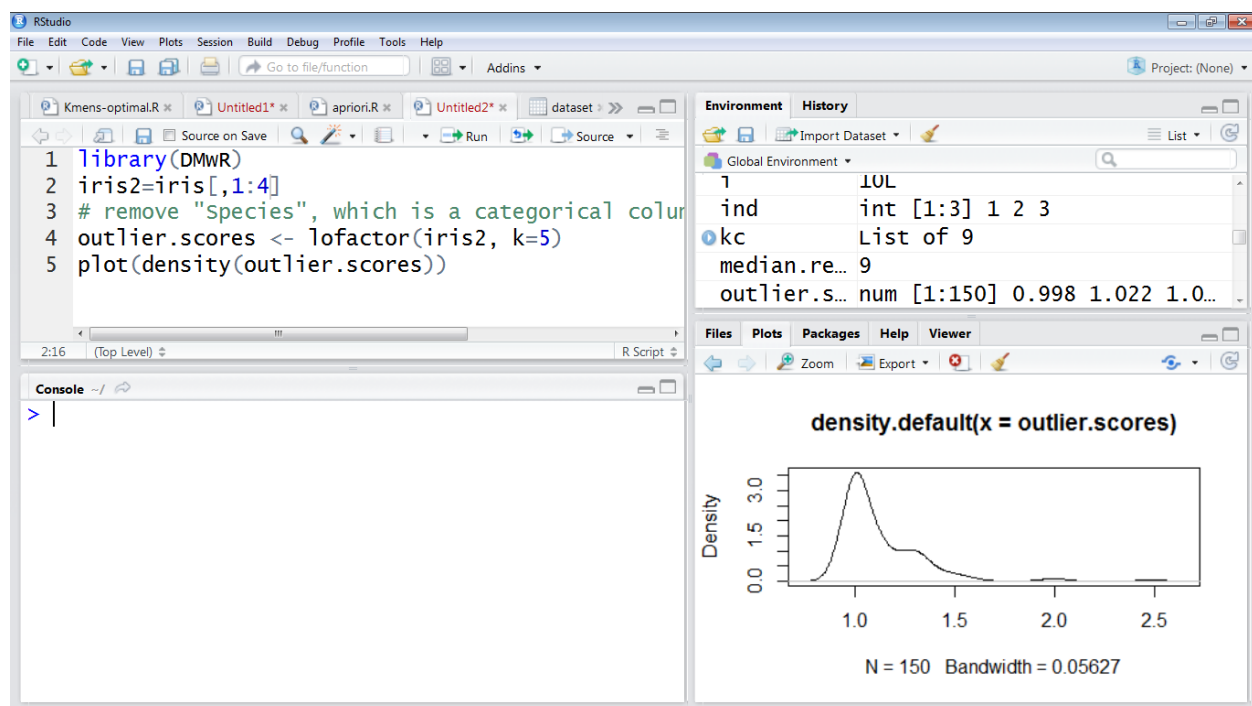
The outliers may be of particular interest, such as in the case of fraud detection, where outliers may indicate fraudulent activity. Thus, outlier detection and analysis is an interesting data mining task, referred to as outlier mining or outlier analysis.

Implementation of Outlier Analysis in R

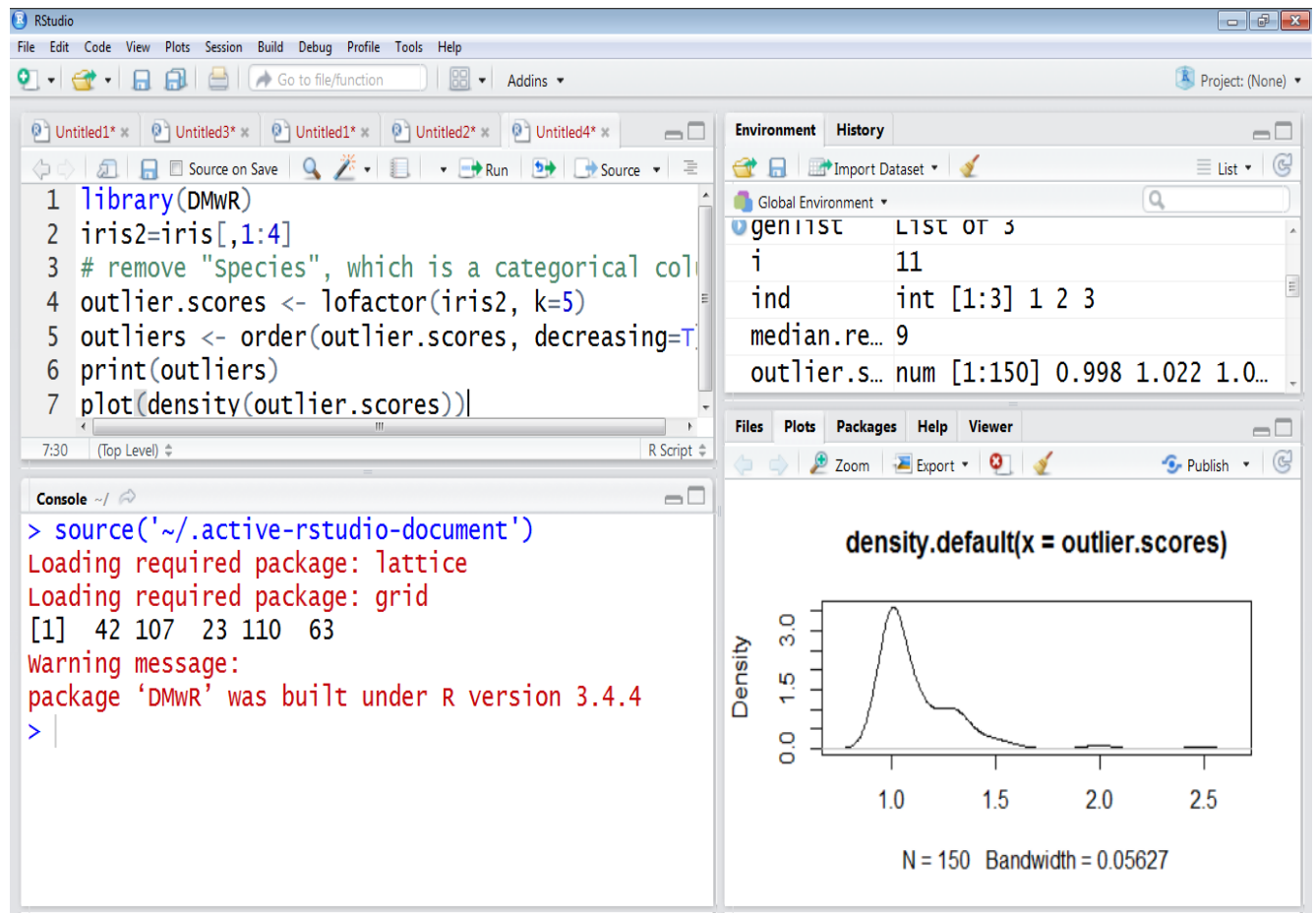
The LOF algorithm

LOF (Local Outlier Factor) is an algorithm for identifying density-based local outliers [Breunig et al., 2000]. With LOF, the local density of a point is compared with that of its neighbors. If the former is significantly lower than the latter (with an LOF value greater than one), the point is in a sparser region than its neighbors, which suggests it be an outlier.

Function `lofactor(data, k)` in packages `DMwR` and `dprep` calculates local outlier factors using the LOF algorithm, where `k` is the number of neighbors used in the calculation of the local outlier factors.

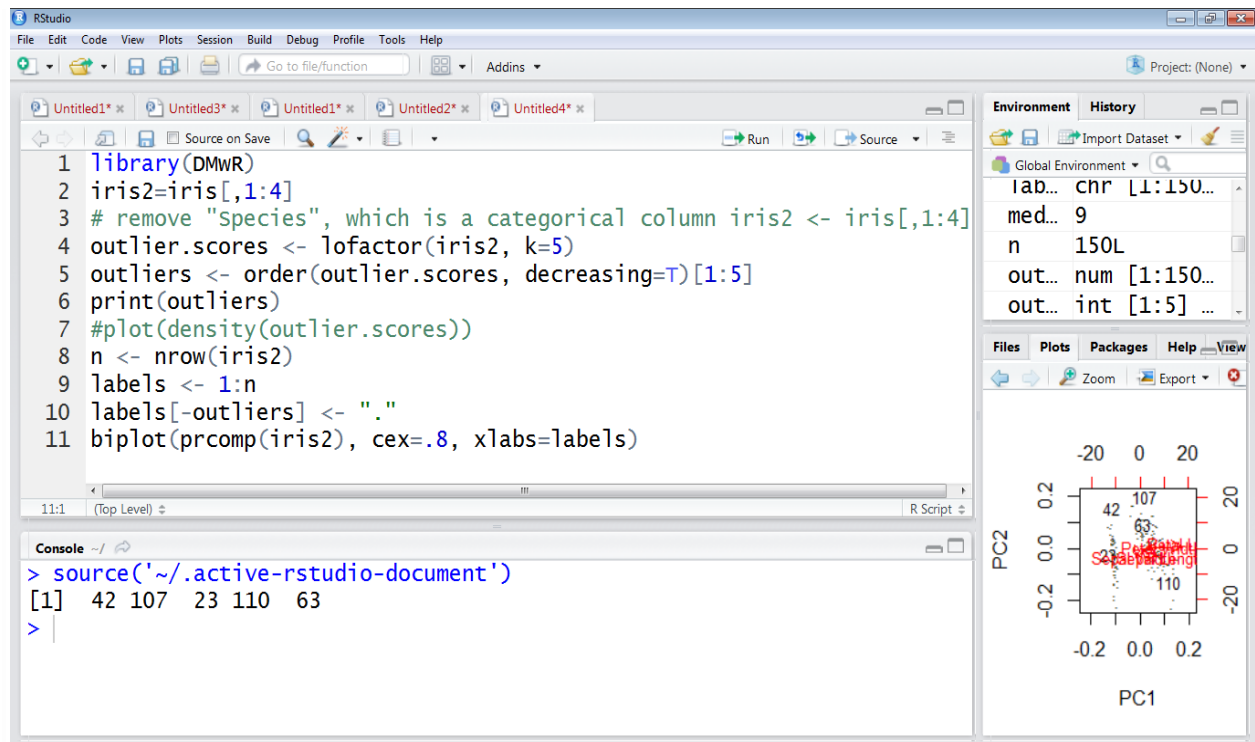


Print the top 5 outliers:



Visualize Outliers with Plots

Next, we show outliers with a biplot of the first two principal components.



Outlier Plot:

