|  |
| --- |
| Report on    *“*Find-S, Candidate Elimination and KNN  Algorithm*”*    *Submitted in partial fulfillment of the requirements for the award of the degree of Master of*  *Technology in Computer Science and Engineering in the course of* ***Machine Learning***  ***(16CSCSPEML)***    Submitted by      **Meghana P**  **1BM17SCS08**        Under the Guidance of  **Dr. B. G. Prasad**  Professor and Head  Department of CSE              Department of Computer Science and Engineering  BMS College of Engineering  P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019  2017-2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **B M S COLLEGE OF ENGINEERING**    **P.O. Box No: 1908 Bull Temple Road Bangalore-560019**    **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**      **ASSESSMENT**      Report on **Machine Learning** (**16CSCSPEML),** “Find-S, Candidate  Elimination and KNN Algorithm”has been successfully completed by **Meghana P (1BM17SCS08)** at B.M.S College ofEngineering in partial fulfilment of the requirements for the 2nd Semester, degree in Master of Technology in Computer Science and Engineering under Visvesvaraya Technological University, Belgaum during academic year 2017-2018.          **Dr. B. G. Prasad Final Marks Awarded**     |  |  | | --- | --- | | **Obtained** | **Total** | |  |  |   Professor and Head  Department of Computer science |

# CONTENTS

1. FIND-S
2. CANDIDATE ELIMINATION
3. K NEAREST NEIGHBOUR

# FIND-S



Hypotheses H: Each hypothesis is a conjunction of constraints on the attributes described above. The constraints may be “?” or “N” (no value is acceptable), or a specific value.

Target concept c: Animal Type: X -> {0,1} (1 if belongs to the class, 0 otherwise) Training examples D: Positive and negative examples of the target function

Determine:

A hypothesis h in H such that h(x) = c(x) for all x in X

The first step is to define the procedure of the Find-S for just one class. So that we can later apply it to each class. The whole algorithm can be reduced to a simpler one: instead of iterating and replacing one value at a time we can just look at the whole column vector of the attribute and then define the resulting hypothesis as follows:

* set value in position “i” of the hypothesis to value *v* if the column i of the dataset has only this value *v*
* set value in position “i” to “?” otherwise

## **FIND-S Algorithm**

|  |
| --- |
| **library**(plyr) findS <- **function**(dset){ apply( dset[,1:ncol(dset)-1],  2, **function**(x) { **if** (length(unique(x)) > 1) "?" **else** unique(x)  })  } |

**Example 1: For Weather data** Step 1: Training examples:

e1 = t(c("Sunny", "Warm", "Normal","Strong","Warm","Same",1)) e2 = t(c("Sunny", "Warm", "High","Strong","Warm","Same",1)) e3 = t(c("Rainy", "Cold", "High","Strong","Warm","Change",0)) e4 = t(c("Sunny", "Warm", "High","Strong","Cool","Change",1))

Step 2: Since we are concerned only about the positve ones, take the subset of those that have “Yes” as the value of the target function:

weather\_dset\_positive\_only = as.data.frame( rbind(e1,e2,e4), stringsAsFactors = FALSE) weather\_dset = as.data.frame( rbind(e1,e2,e3, e4), stringsAsFactors = FALSE)

Step 3: Testing

weather\_concept = findS(weather\_dset\_positive\_only) weather\_concept

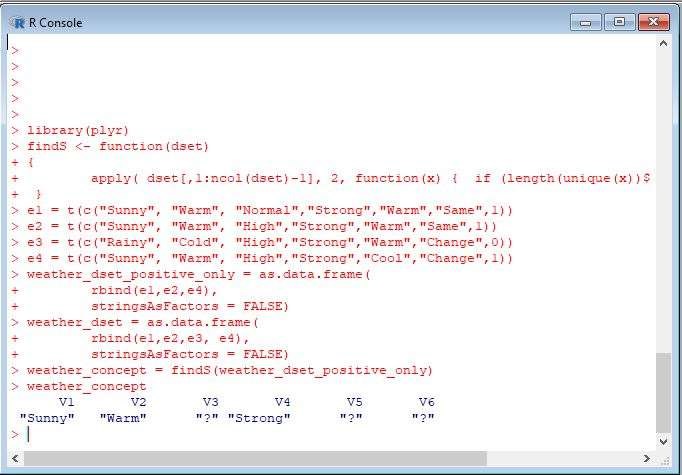


Fig 1. Result of Find-S algorithm for Weather data example

## **Example 2: For Animal data**

Download zoo.csv file with following attributes

Animal\_name, hair, feathers, eggs, milk, airborne, aquatic, predator, toothed, backbone , breathes, venomous, fins, legs, tail, domestic, catsize.

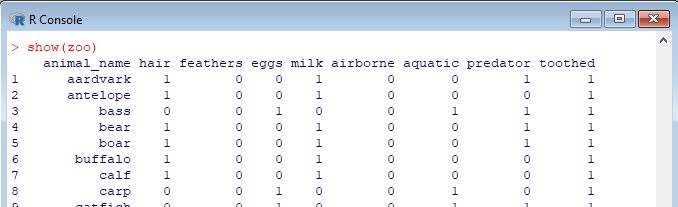


Fig 2. Input file zoo.csv

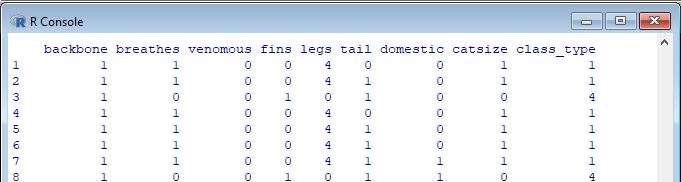


Fig 2. Input file zoo.csv

Step 1:

|  |
| --- |
| zoo <- read.csv("C:/Users/Nikitaa/Desktop/ML/zoo.csv", header=FALSE, stringsAsFactors=FALSE)  cnames = c("name", "hair", "feathers", "eggs", "milk", "airborne","aquatic"," predator","toothed","backbone","breathes", "venomous", "fins", "legs", "tail"  , "domestic","catsize", "type") colnames(zoo) = cnames |

Step 2:

h\_set = ddply(zoo, .(type), findS) h\_set

### Output

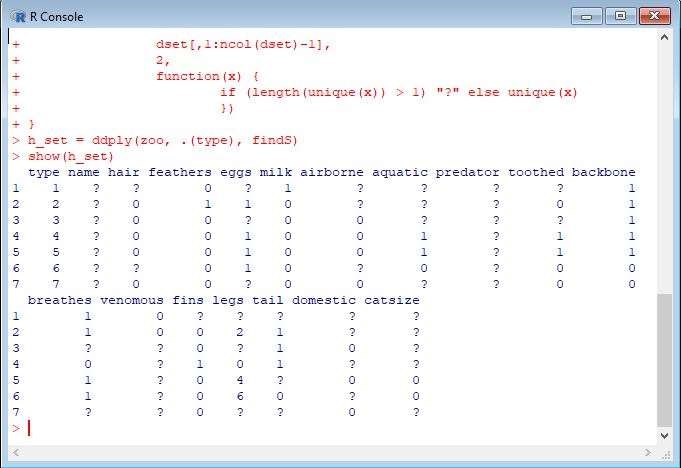
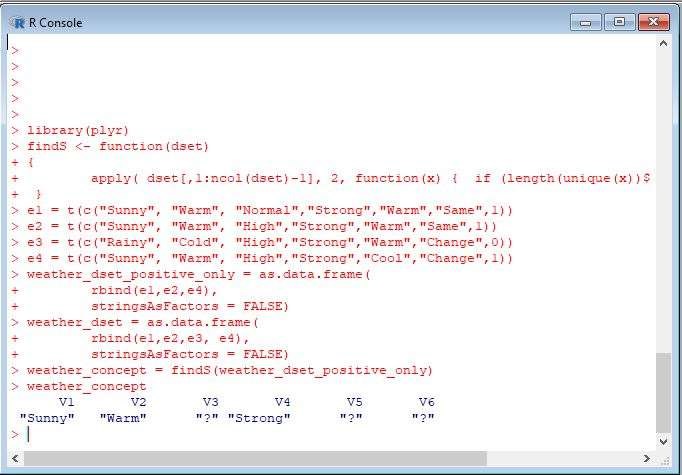


Fig 3. Result of Find-S algorithm for animal data set

# CANDIDATE ELIMINATION



Start with FindS to define the most specific hypothesis. It is equivalent to the situation when we have only the positive examples at the beginning and since the algorithms doesn’t specifiy it cannot happen we assume that it can (according to probability theory). So the findS results is basically the S boundary because:

* it is consistent with all positive examples we’ve ‘seen’ (so basically all positive examples in the dataset)
* minimal generalizations according to findS are added just to satisfy the new examples we see
* the only member in G (which is all “?” at the beginning) in this case is by definition more general than anything we have in S. The border case is when our S collapses to all “?”, but this means that the concept is not teachable.

Another consequence is that our S boundary will always contain just one hypothesis So we can treat it’s a vector. This is just the implementation bonus. The other tricks I used also speed up the learning significantly.

**Example** : Zoo data set

## Step 1: Simple preprocessing

For each class perform one-vs-all classification. *preprocess* function basically labels the examples.

preprocess <- **function**(dset,class\_id) { dset = as.matrix(dset) svec = dset[,ncol(dset)]==class\_id dset[svec,ncol(dset)] = 1 dset[!svec,ncol(dset)] = 0 positive = dset[svec,] negative = dset[!svec,] list(positive=positive,negative=negative)

}

Step 2: Checking the consistency

To check if the example is consistent with the hypothesis I wrote a general function that checks if the hypothesis accepts the positive example and rejects the negative one. This little function is really important since it allows us to fail early and notice that there is something wrong with the instace space. This function is used to check if S is still valid.

|  |
| --- |
| consistent\_he <- **function**(hypothesis, example) { true\_label = example[length(example)] d = example[1:length(example)-1] satisfies = all(hypothesis == d | hypothesis == "?")  TP = satisfies && true\_label == 1 TN = !satisfies && true\_label == 0 **return**(TP || TN)  } |

Step 3:Minimal specializations

This function finds the minimal specializations for the given negative example and the hypotheses that let him in. Although it’s possible in theory to generate all possible combinations consistent with the example etc, in fact we can significantly reduce computational time if we add wisely when generating new hypotheses. Here are my steps to get specializations:

* find positions for which the following is true
  + they cause the hypothesis from the general boundary lets the negative example in
  + the s-hypothesis has some value in them and it’s not equal to the value in the negative example
* stick the values from the s-hypothesis to these slots in the g-hypothesis one a time (creating a new hypothesis each time)

|  |
| --- |
| get\_specializations <- **function**(S,Gn,d) { **if** (ncol(Gn)!=length(d)) { d = d[1:ncol(Gn)]} s = S[1,] *#convert to vector* n = nrow(Gn) result = Gn[F,] **for** (i **in** seq(1:n)) { g = Gn[i,] *# get a vector* indices = which((g=="?" | g==d) & (s != d & s != "?")) **for** (i **in** indices) { nh = g, nh[i] = s[i] result = rbind(result,nh)  }  } **return**(result)  } |

Step 4: More general

This is a simple function for strict inequality of the hypothesis

more\_general <- **function**(hypothesis1,hypothesis2) { all( hypothesis1[hypothesis1 != hypothesis2] == "?") && any(hypothesis1 != hypothesis2)

}

Step 5: Pruning the G boundary

pruneG <- **function**(G) { selmatrix = t(apply(G, 1, **function**(h) {apply(G,1,more\_general,h)})) selvec = !apply(selmatrix,1,any) subset(G,selvec)

}

Step 6: Final implementation

This function combines everything together and allows us to finally apply the algorithms to our dataset.

|  |
| --- |
| candidate\_elimination <- **function**(dset, class\_id) { dset = preprocess(dset,class\_id)  S = as.matrix(t(findS(dset$positive)))  G = matrix(data=rep("?",ncol(S)),nrow=1, byrow=TRUE) colnames(G) = colnames(S) n\_negative = nrow(dset$negative) **for** (i **in** seq(1:n\_negative)) { d = dset$negative[i,] **if** (consistent\_he(S,d)) { consistent\_hypotheses = apply(G,1,consistent\_he,d)  Gn = subset(G,!consistent\_hypotheses) G = subset(G, consistent\_hypotheses) **if** (nrow(Gn) > 0) {  Gn = get\_specializations(S,Gn,d)  }  G = rbind(G,Gn) G = pruneG(G)  } **else** { print(d) message("Failed: concept can't be learned") **return**('FAIL')  }  } **return**(list(G=(unique(G)),S=S))  } |

Step 7: Reconstructing the version space

|  |
| --- |
| build\_version\_space <- **function**(S,G) { s = S[1,] *# vector* n = nrow(G) V = S[F,] **for** (i **in** seq(1:n)) { g = G[i,] pos = which(g!=s & s!="?") m = length(pos) **for** (j **in** seq(1:(m-1))) { combs = combn(pos,j) **for** (k **in** seq(1:ncol(combs))) { nh = s selvec = combs[,k] nh[selvec] = "?" V = rbind(V,nh)  }  }  } **return**(unique(V))  } |

Step 8: Putting everything together

class\_id = 1 res = candidate\_elimination(zoo,class\_id) V = build\_version\_space(res$S,res$G) results = list(G= res$G, S = res$S, V = V)

V

### Output

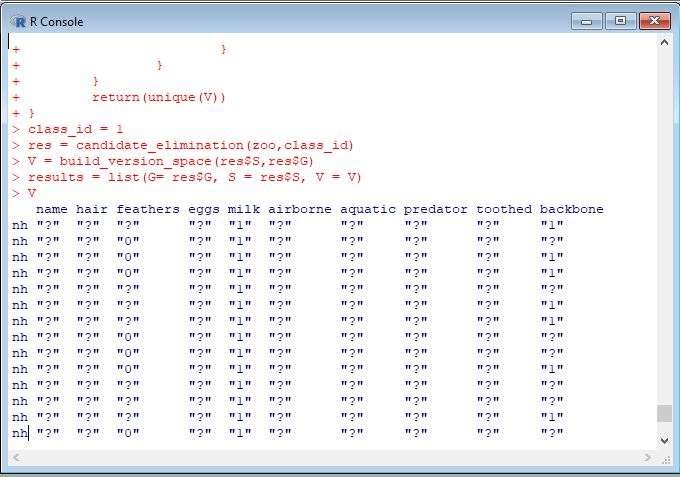


Fig 4. Result of candidate elimination on zoo data set

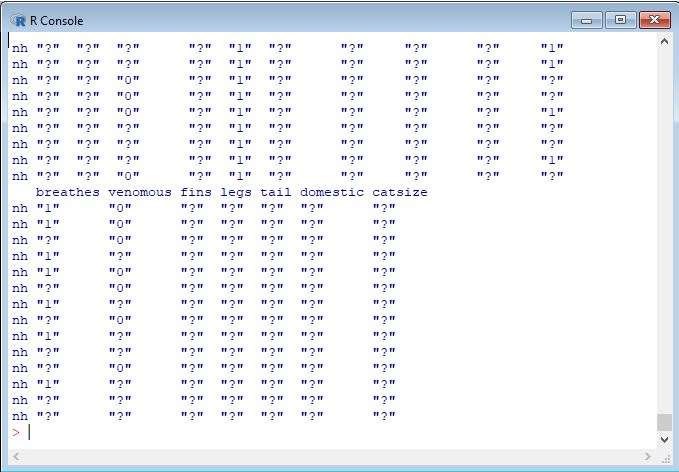


Fig 4. Result of candidate elimination on zoo data set

# K nearest Neighbour



## Step 1- Data collection

We will use a data set of 100 patients to implement the knn algorithm and thereby interpreting results .The data set has been prepared keeping in mind the results which are generally obtained from DRE. The data set consists of 100 observations and 10 variables which are as follows:

1. Radius
2. Texture
3. Perimeter
4. Area
5. Smoothness
6. Compactness
7. Symmetry
8. Fractal dimension

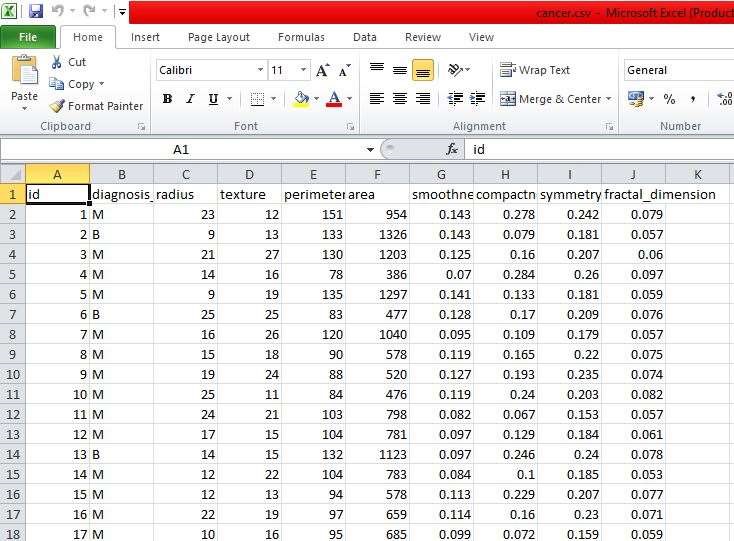
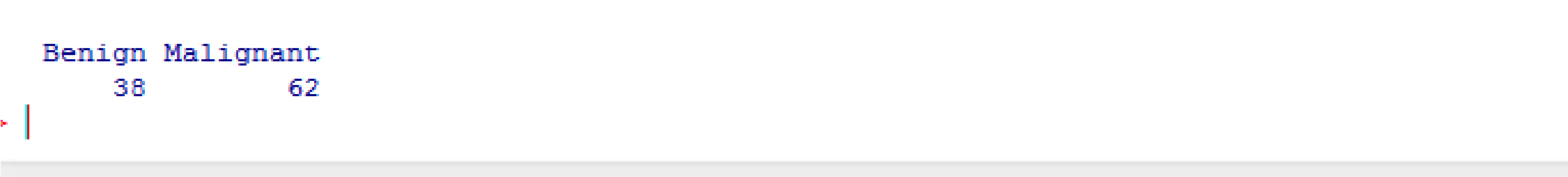
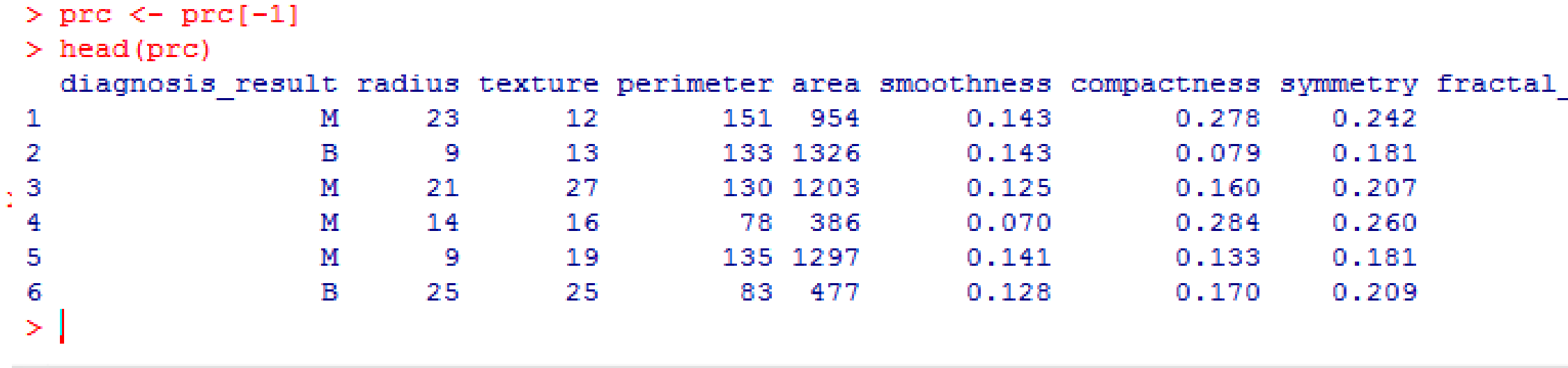


Fig 5. Input file for KNN i.e File containing cancer details

## Step 2- Preparing and exploring the data

setwd("C:/Users/Nikitaa/Desktop/ML") prc <- read.csv("cancer.csv",stringsAsFactors = FALSE) stringsAsFactors = FALSE str(prc)

prc <- prc[-1]

In case we wish to rename B as ”Benign” and M as “Malignant” and see the results in the percentage form, we may write as:

prc$diagnosis <- factor(prc$diagnosis\_result, levels = c("B", "M"), labels = c("Benig n", "Malignant")) round(prop.table(table(prc$diagnosis)) \* 100, digits = 1)

Normalizing numeric data

This feature is of paramount importance since the scale used for the values for each variable might be different. The best practice is to normalize the data and transform all the values to a common scale.

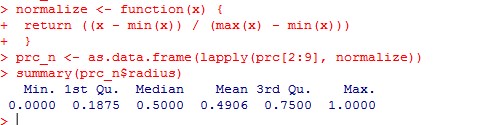
normalize <- function(x) {return ((x - min(x)) / (max(x) - min(x))) }

Instead of normalizing each of the 8 individual variables we use:

prc\_n <- as.data.frame(lapply(prc[2:9], normalize))

The first variable in our data set is ‘diagnosis\_result’ which is not numeric in nature. So, we start from 2nd variable. The function lapply() applies normalize() to each feature in the data frame. The final result is stored to prc\_n data frame using as.data.frame() function

summary(prc\_n$radius)



Creating training and test data set

The kNN algorithm is applied to the training data set and the results are verified on the test data set.

For this, we would divide the data set into 2 portions in the ratio of 65: 35 (assumed) for the training and test data set respectively. You may use a different ratio altogether depending on the business requirement!

We shall divide the prc\_n data frame into prc\_train and prc\_test data frames

prc\_train <- prc\_n[1:65,]

prc\_test <- prc\_n[66:100,]

## Step 3 – Training a model on data

The knn () function needs to be used to train a model for which we need to install a package ‘class’. The knn() function identifies the k-nearest neighbors using Euclidean distance where k is a user-specified number. Now we are ready to use the knn() function to classify test data

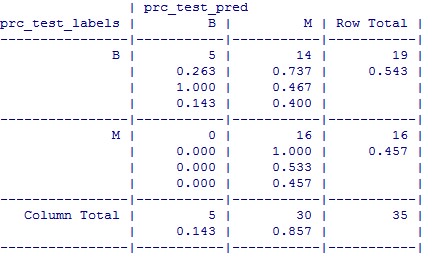
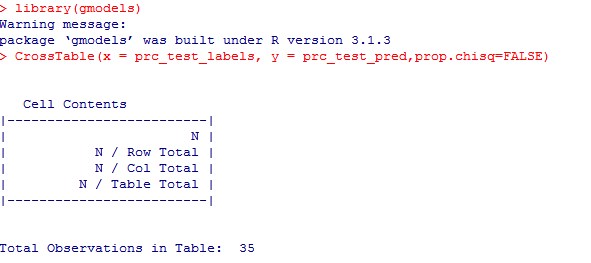
prc\_test\_pred <- knn(train = prc\_train, test = prc\_test,cl = prc\_train\_labels, k=10)

The value for k is generally chosen as the square root of the number of observations.

knn() returns a factor value of predicted labels for each of the examples in the test data set which is then assigned to the data frame prc\_test\_pred

Step 4 – Evaluate the model performance

We have built the model but we also need to check the accuracy of the predicted values in prc\_test\_pred as to whether they match up with the known values in prc\_test\_labels. To ensure this, we need to use the CrossTable() function available in the package ‘gmodels’.



The test data consisted of 35 observations. Out of which 5 cases have been accurately predicted (TN->True Negatives) as Benign (B) in nature which constitutes 14.3%. Also, 16 out of 35 observations were accurately predicted (TP> True Positives) as Malignant (M) in nature which constitutes 45.7%. Thus a total of 16 out of 35 predictions where TP i.e, True Positive in nature.

There were no cases of False Negatives (FN) meaning no cases were recorded which actually are malignant in nature but got predicted as benign. The FN’s if any poses a potential threat for the same reason and the main focus to increase the accuracy of the model is to reduce FN’s.

There were 14 cases of False Positives (FP) meaning 14 cases were actually benign in nature but got predicted as malignant.

The total accuracy of the model is 60 %( (TN+TP)/35) which shows that there may be chances to improve the model performance