Final Assignment Models Predictors

Eduard Romero / Carlos Bejarano / Juan Cepeda

2023-05-27

1. Problem definition.

Mobile robotics is a popular solution for exploration of hostile environments (such as toxic or radioactive environments) where a direct human intervention is not possible. In this project it is asked that each team implements a robotic explorer and simulates 3 different environments.

- 1.1 A total of 3 different environments needs to be simulated. Each environment needs to provide at least 3 conditions that can be sensed by the robot.
- 1.1.2 The robot needs to be able of moving from one environment to another.
- 1.1.3 Configuration of the environments (order) must be interchangeable.
- 1.1.4 The robot needs to acquire 3 or more sensor signals that can be use as predictors for supervised algorithms.

1.1 Controlled environments.

- Cold Room.
- Hot Room.
- Toxic Room.

1.1.2 Robot characteristics.

The main features of our robot are as follows:

- MCU Arduino.
- Micromotors DC.
- PCB.
- Battery LIPO 7.4V 300mAh.
- Driver motor TB6612FNG.
- Bluetooth HC06.

1.1.3 Configuration enviroments.

The configuration of our 3 environments will be placed in cascade form one after the other, where our robot will take 50 samples for each environment, each of them will be made of cardboard boxes and conditioned for each situation mentioned.

1.1.4 Configuration Sensors.

Our robot has 3 analog sensors:

- Sensor for air quality measurement MQ135.
- Sensor Humidity DHT11.
- Sensor Temperature LM35(DHT11).

The prediction methods will be as follows :

- KNN.
- Logistic regression.
- Decision tree.
- Lasso regression.

2. Arduino code program.

Code Source on repository GitHub

3. Methods for prediction.

- 3.1 KNN without preprocessing: K-Nearest Neighbors (KNN) is a non-parametric classification algorithm. It makes predictions based on the majority class of the K nearest neighbors in the feature space. Without preprocessing, KNN uses the raw data as input, without any transformation or scaling.
- 3.2 KNN with preprocessing: KNN can benefit from preprocessing techniques such as feature scaling, normalization, or dimensionality reduction. Preprocessing helps to improve the performance and accuracy of KNN by ensuring that all features are on a similar scale or reducing the dimensionality of the data.
- 3.3 KNN Grid: KNN Grid is a technique that helps to determine the optimal value of K in KNN by performing a grid search. It involves training and evaluating multiple KNN models with different values of K and selecting the value that produces the best performance or accuracy on the validation set.
- 3.4 Logistic Regression: Logistic Regression is a supervised learning algorithm used for binary classification problems. It models the relationship between the independent variables (features) and the probability of a certain outcome using the logistic function. It estimates the parameters of the logistic function using maximum likelihood estimation.
- 3.5 Decision Tree: Decision Tree is a supervised learning algorithm that builds a tree-like model for classification or regression. It splits the data based on features at each node and makes predictions by traversing the tree from the root to the leaf nodes. It selects the best feature to split based on certain criteria such as information gain or Gini index.

-3.6 Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It creates a set of decision trees using bootstrapped samples of the data and random feature subsets. The final prediction is made by aggregating the predictions of individual trees.

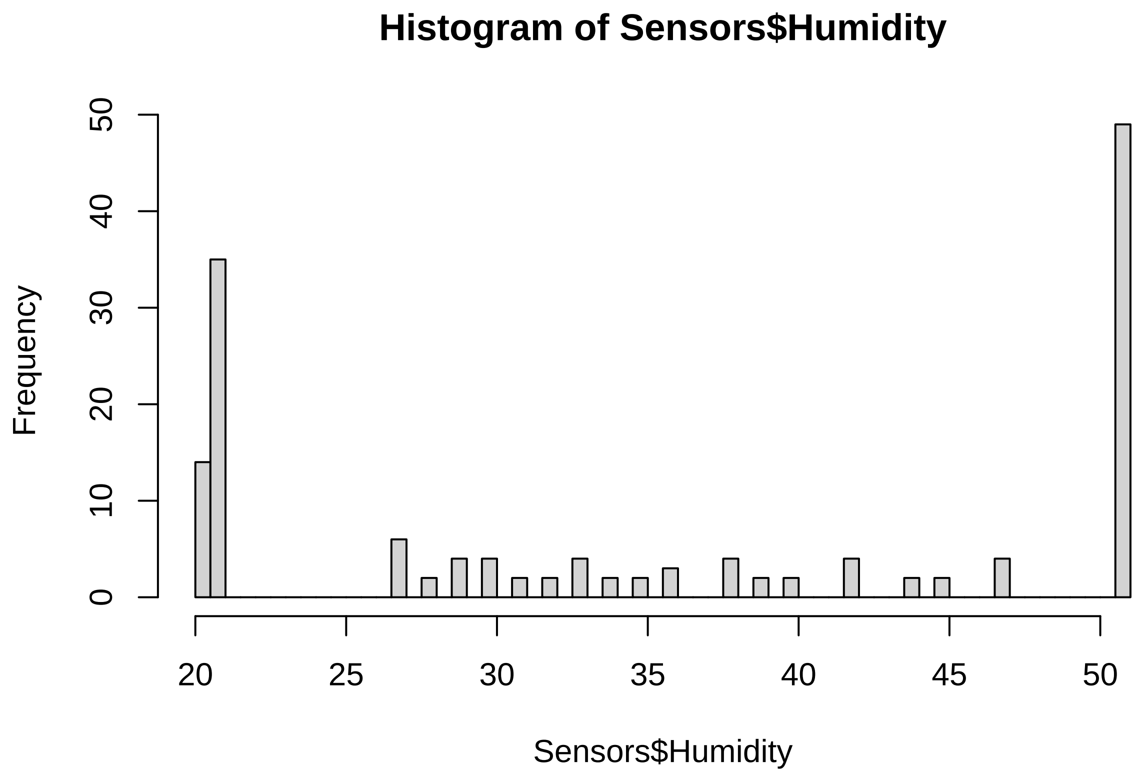
-3.7 Naive Bayes: Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem. It assumes that the features are conditionally independent given the class label. Naive Bayes calculates the probability of each class and predicts the class with the highest probability.

```r
# Import the librarys
library (tidyverse)
library (caret)
library(psych)
library(ggplot2)
library(MASS)
library(nnet)
library(rpart)
library(rpart.plot)
library(randomForest)
library(e1071)
library(tm)
library(naivebayes)
```

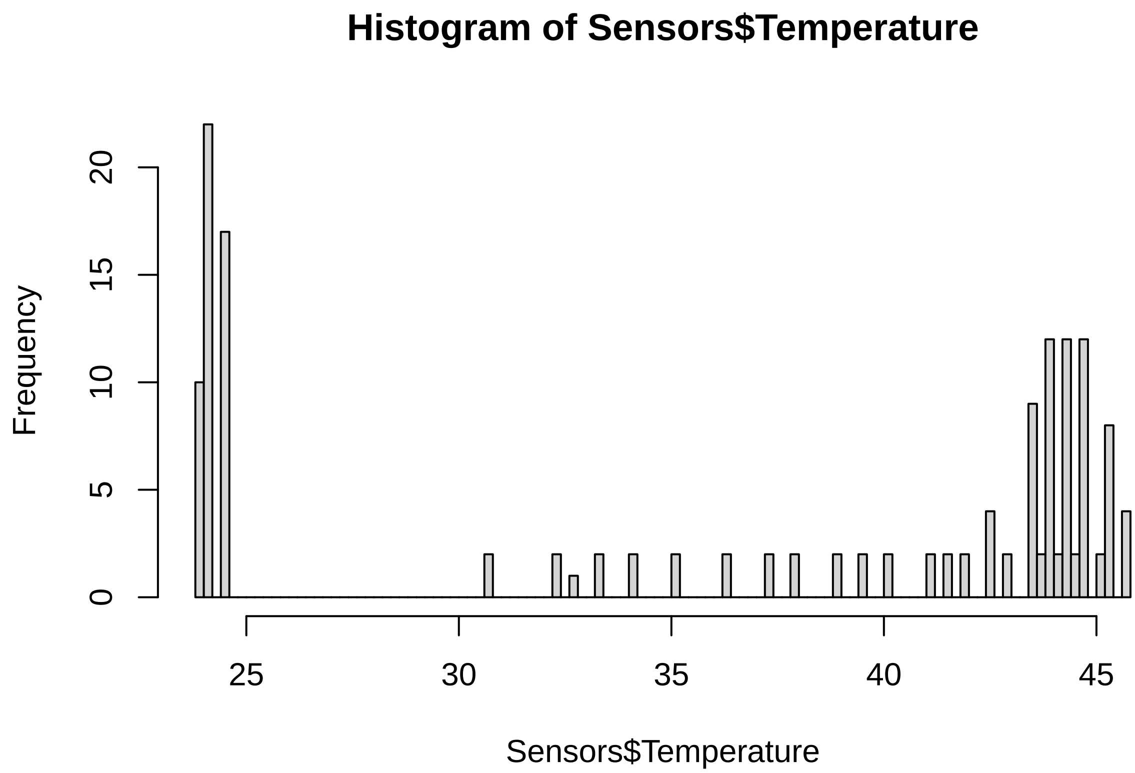
```r
# Obtain the current folder path and its parent folder path

folder <- dirname(rstudioapi::getSourceEditorContext()$path)
parentFolder <- dirname(folder)

#Read CSV File for trainning

Sensors <- read_csv(file = paste0(parentFolder, "/Datasets/Train_data.csv")) %>% as.data.frame()
#Read CSV File for Predict

DataTest <- read_csv(file = paste0(parentFolder, "/Datasets/Model.csv")) %>% as.data.frame()

# Give our a summary for variables Humidity,Temperatura,PPM,Room

summary(Sensors)
```

```
##     Humidity      Temperature         PPM              Room          
##  Min.   :20.00   Min.   :23.80   Min.   :  9.18   Length:149        
##  1st Qu.:21.00   1st Qu.:24.50   1st Qu.: 17.64   Class :character  
##  Median :34.00   Median :41.60   Median : 19.95   Mode  :character  
##  Mean   :35.68   Mean   :36.28   Mean   :160.63                     
##  3rd Qu.:51.00   3rd Qu.:44.40   3rd Qu.:337.11                     
##  Max.   :51.00   Max.   :45.70   Max.   :575.00                     
```

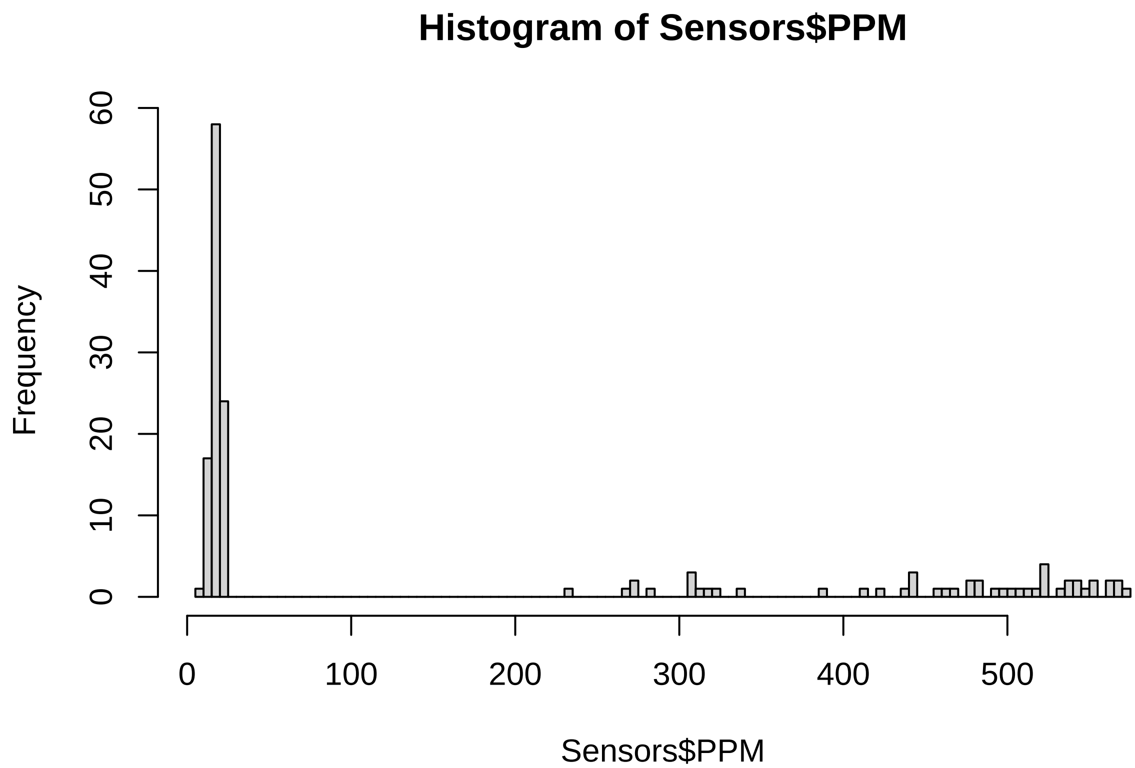
```r
# Histogram of the linear model Humidity

hist(Sensors$Humidity,breaks = 100)
```



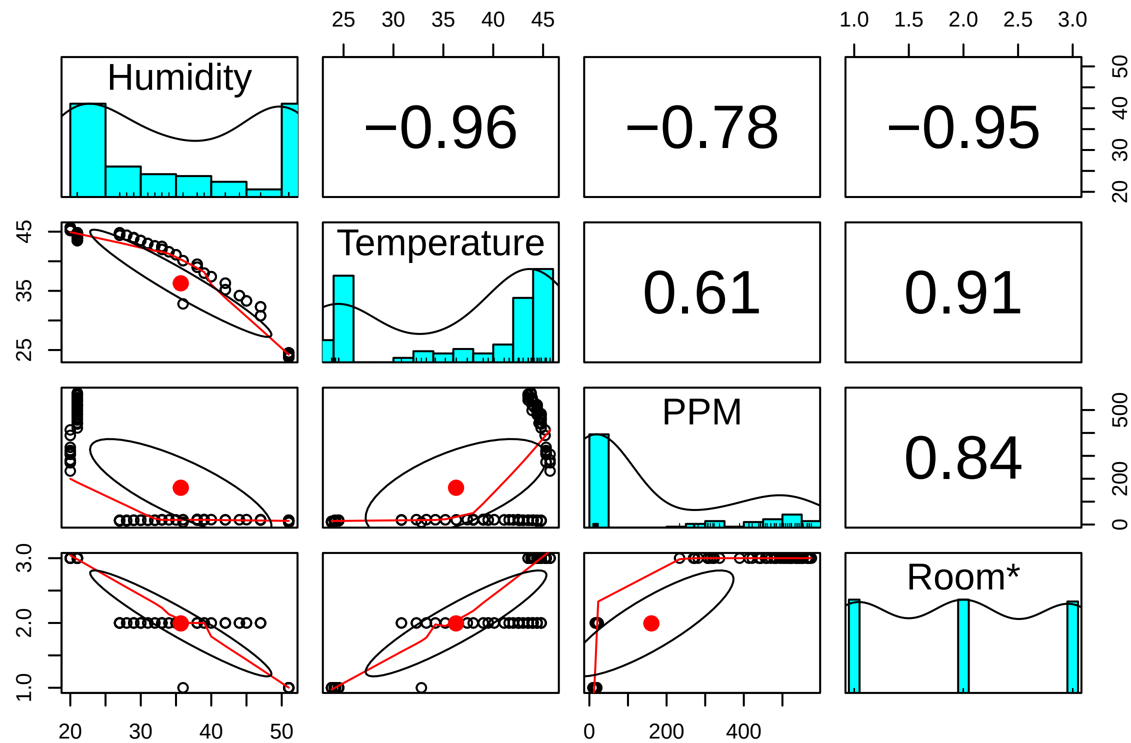

```r
# Histogram of the linear model Temperature

hist(Sensors$Temperature,breaks = 100)
```




```r
# Histogram of the linear model PPM

hist(Sensors$PPM,breaks = 100)
```




```
pairs.panels(Sensors[c("Humidity",

                       "Temperature",

                       "PPM",

                       "Room")]

              ,pch=21, bg=c("red","green3","blue", "orange")[unclass(Sensors$Room)])
```




```r
predictors <- colnames(Sensors)[-3]

sample.index <- sample(1:nrow(Sensors)
                       ,nrow(Sensors)*0.3
                       ,replace = F)

train.data <- Sensors[sample.index,c(predictors,"Room"),drop=F]
test.data <- Sensors[-sample.index,c(predictors,"Room"),drop=F]

# Use 10-Fold cross-validation for all methods

Model <-trainControl(method="cv",number=10)

# Train Model Knn without processing
Model1 <- train(Room~.,data = Sensors,method="knn",trControl=Model)
Model1
```

```
## k-Nearest Neighbors
##
## 149 samples
##   3 predictor
##   3 classes: 'Cold', 'Hot', 'Toxic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 135, 134, 134, 134, 134, 134, ...
## Resampling results across tuning parameters:
```

```
##
##   k  Accuracy   Kappa
##   5  0.9933333  0.99
##   7  0.9933333  0.99
##   9  0.9933333  0.99
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```r
# Predict Model Knn without processing

Predict1 <- predict(Model1,newdata=DataTest,)
Predict1
```

```
##  [1] Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold
## [13] Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold
## [25] Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Hot
## [37] Hot   Hot   Hot   Hot   Cold  Hot   Cold  Hot   Hot   Hot   Hot   Hot
## [49] Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot
## [61] Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic
## [73] Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic
## [85] Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic
## Levels: Cold Hot Toxic
```

```r
# Train Model Knn with processing

Model2 <- train(Room~.,data = Sensors,method="knn",preProcess=c("center","scale"),trControl=Model)
Model2
```

```
## k-Nearest Neighbors
##
## 149 samples
##   3 predictor
##   3 classes: 'Cold', 'Hot', 'Toxic'
##
## Pre-processing: centered (3), scaled (3)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 134, 134, 135, 134, 134, 134, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9928571  0.9892308
##   7  0.9928571  0.9892308
##   9  0.9861905  0.9792308
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```r
# Predict Model Knn with processing

Predict2 <- predict(Model2,newdata =DataTest,)
Predict2
```

```
##  [1] Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold
## [16] Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold
## [31] Cold Cold Cold Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
## [46] Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
```

```
## [61] Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
## [76] Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
## [91] Hot  Hot  Hot  Hot
## Levels: Cold Hot Toxic
```

```
# Train Model Knn Grid

knnGrid <- expand.grid(k=c(1,5,10,30,100))
Model3 <- train(Room~.,data = Sensors,method="knn",preProcess=c("center","scale"),tuneGrid=knnGrid,trCon
Model3
```

```
## k-Nearest Neighbors
##
## 149 samples
##   3 predictor
##   3 classes: 'Cold', 'Hot', 'Toxic'
##
## Pre-processing: centered (3), scaled (3)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 135, 134, 134, 134, 134, 134, ...
## Resampling results across tuning parameters:
##
##   k    Accuracy   Kappa
##     1  0.9933333  0.9900000
##     5  0.9933333  0.9900000
##    10  0.9933333  0.9900000
##    30  0.9661905  0.9492308
##   100  0.4157143  0.1223256
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 10.
```

```
# Predict model Knn Grid

Predict3 <- predict(Model3,newdata = DataTest,)
Predict3
```

```
##  [1] Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold
## [16] Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold Cold
## [31] Cold Cold Cold Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
## [46] Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
## [61] Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
## [76] Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot  Hot
## [91] Hot  Hot  Hot  Hot
## Levels: Cold Hot Toxic
```

```
# Train Model Logistic regression

Model4 <- multinom(Room~.,data=Sensors,iter=500)
```

```
## # weights:  15 (8 variable)
## initial  value 163.693231
## iter  10 value 9.157736
## iter  20 value 5.793704
## iter  30 value 0.043440
## iter  40 value 0.000867
## iter  50 value 0.000750
```

```
## iter  60 value 0.000514
## iter  70 value 0.000474
## iter  80 value 0.000426
## iter  90 value 0.000340
## iter 100 value 0.000280
## final  value 0.000280
## stopped after 100 iterations
```

```
summary(Model4)
```

```
## Call:
## multinom(formula = Room ~ ., data = Sensors, iter = 500)
##
## Coefficients:
##       (Intercept)  Humidity Temperature      PPM
## Hot   -15.7849519 -2.647689   1.3048461 5.736593
## Toxic   0.9831083 -3.090925   0.7747373 5.935726
##
## Std. Errors:
##       (Intercept)  Humidity Temperature       PPM
## Hot     10.374726  18.77075    10.92369  42.69913
## Toxic    4.853577 149.92168   105.95862 448.78078
##
## Residual Deviance: 0.0005604915
## AIC: 16.00056
```

```
# Predict Model Logistic regression

Predict4 <- predict(Model4,newdata=DataTest,)
Predict4
```

```
##  [1] Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Cold  Hot
## [13] Cold  Cold  Cold  Cold  Hot   Cold  Cold  Cold  Cold  Cold  Hot   Cold
## [25] Hot   Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold
## [37] Hot   Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Cold  Hot
## [49] Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot   Hot
## [61] Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic
## [73] Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic
## [85] Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic Toxic
## Levels: Cold Hot Toxic
```

```
prediction<-DataTest$Predict4 <- c(Predict4)
confusionMatrix(prediction,DataTest$Predict4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Cold Hot Toxic
##      Cold    32   0     0
##      Hot      0  28     0
##      Toxic    0   0    34
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9615, 1)
```

```
##     No Information Rate : 0.3617
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Cold Class: Hot Class: Toxic
## Sensitivity               1.0000     1.0000       1.0000
## Specificity               1.0000     1.0000       1.0000
## Pos Pred Value            1.0000     1.0000       1.0000
## Neg Pred Value            1.0000     1.0000       1.0000
## Prevalence                0.3404     0.2979       0.3617
## Detection Rate            0.3404     0.2979       0.3617
## Detection Prevalence      0.3404     0.2979       0.3617
## Balanced Accuracy         1.0000     1.0000       1.0000
```

```
# Train Model Decision tree

CartGrid = expand.grid(maxdepth=c(1,5,10,20))
Model5 <- train(Room~.,data=Sensors,method="rpart2",trControl=Model,tuneGrid=CartGrid)
Model5
```

```
## CART
##
## 149 samples
##   3 predictor
##   3 classes: 'Cold', 'Hot', 'Toxic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 134, 134, 134, 134, 134, 134, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##    1        0.9928571  0.9892308
##    5        0.9928571  0.9892308
##   10        0.9928571  0.9892308
##   20        0.9928571  0.9892308
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 1.
```

```
Model3$finalModel
```

```
## 10-nearest neighbor model
## Training set outcome distribution:
##
##  Cold   Hot Toxic
##    50    50    49
```

```
# Predict Model Decision tree

Predict5 <- predict(Model5,newdata=DataTest,type='prob')
```

```
Predict5
```

```
##    Cold Hot Toxic
## 1   0.5 0.5     0
## 2   0.5 0.5     0
## 3   0.5 0.5     0
## 4   0.5 0.5     0
## 5   0.5 0.5     0
## 6   0.5 0.5     0
## 7   0.5 0.5     0
## 8   0.5 0.5     0
## 9   0.5 0.5     0
## 10  0.5 0.5     0
## 11  0.5 0.5     0
## 12  0.5 0.5     0
## 13  0.5 0.5     0
## 14  0.5 0.5     0
## 15  0.5 0.5     0
## 16  0.5 0.5     0
## 17  0.5 0.5     0
## 18  0.5 0.5     0
## 19  0.5 0.5     0
## 20  0.5 0.5     0
## 21  0.5 0.5     0
## 22  0.5 0.5     0
## 23  0.5 0.5     0
## 24  0.5 0.5     0
## 25  0.5 0.5     0
## 26  0.5 0.5     0
## 27  0.5 0.5     0
## 28  0.5 0.5     0
## 29  0.5 0.5     0
## 30  0.5 0.5     0
## 31  0.5 0.5     0
## 32  0.5 0.5     0
## 33  0.5 0.5     0
## 34  0.5 0.5     0
## 35  0.5 0.5     0
## 36  0.5 0.5     0
## 37  0.5 0.5     0
## 38  0.5 0.5     0
## 39  0.5 0.5     0
## 40  0.5 0.5     0
## 41  0.5 0.5     0
## 42  0.5 0.5     0
## 43  0.5 0.5     0
## 44  0.5 0.5     0
## 45  0.5 0.5     0
## 46  0.5 0.5     0
## 47  0.5 0.5     0
## 48  0.5 0.5     0
## 49  0.5 0.5     0
## 50  0.5 0.5     0
## 51  0.5 0.5     0
```

```
## 52  0.5 0.5     0
## 53  0.5 0.5     0
## 54  0.5 0.5     0
## 55  0.5 0.5     0
## 56  0.5 0.5     0
## 57  0.5 0.5     0
## 58  0.5 0.5     0
## 59  0.5 0.5     0
## 60  0.5 0.5     0
## 61  0.5 0.5     0
## 62  0.5 0.5     0
## 63  0.5 0.5     0
## 64  0.5 0.5     0
## 65  0.5 0.5     0
## 66  0.5 0.5     0
## 67  0.5 0.5     0
## 68  0.5 0.5     0
## 69  0.5 0.5     0
## 70  0.5 0.5     0
## 71  0.5 0.5     0
## 72  0.5 0.5     0
## 73  0.5 0.5     0
## 74  0.5 0.5     0
## 75  0.5 0.5     0
## 76  0.5 0.5     0
## 77  0.5 0.5     0
## 78  0.5 0.5     0
## 79  0.5 0.5     0
## 80  0.5 0.5     0
## 81  0.5 0.5     0
## 82  0.5 0.5     0
## 83  0.5 0.5     0
## 84  0.5 0.5     0
## 85  0.5 0.5     0
## 86  0.5 0.5     0
## 87  0.5 0.5     0
## 88  0.5 0.5     0
## 89  0.5 0.5     0
## 90  0.5 0.5     0
## 91  0.5 0.5     0
## 92  0.5 0.5     0
## 93  0.5 0.5     0
## 94  0.5 0.5     0
```

```r
# Train Model Random Forests

set.seed(2018)
Sensors$Room=factor(Sensors$Room)
randomf <-Sensors[complete.cases(Sensors),]
training.ids<-createDataPartition(Sensors$Room,p=0.7,list = F)
modrf <- randomForest(x=Sensors[training.ids,1:3],
                    y=Sensors[training.ids,4],
                    ntree =500,
                    keep.forest = TRUE)
```

```r
# Predict Model Random Forest

Predict6 <- predict(modrf,newdata=DataTest,)
prediction1 <- DataTest$Predict6 <- c(Predict6)
confusionMatrix(prediction1,DataTest$Predict6)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Cold Hot Toxic
##      Cold    37   0     0
##      Hot      0  37     0
##      Toxic    0   0    20
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9615, 1)
##     No Information Rate : 0.3936
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Cold Class: Hot Class: Toxic
## Sensitivity               1.0000     1.0000       1.0000
## Specificity               1.0000     1.0000       1.0000
## Pos Pred Value            1.0000     1.0000       1.0000
## Neg Pred Value            1.0000     1.0000       1.0000
## Prevalence                0.3936     0.3936       0.2128
## Detection Rate            0.3936     0.3936       0.2128
## Detection Prevalence      0.3936     0.3936       0.2128
## Balanced Accuracy         1.0000     1.0000       1.0000
```

```r
# train Model Naivebayes

set.seed(2018)
t.idsl <- createDataPartition(Sensors$Room,p=0.7,list=F)

Model7 <- naiveBayes(Room~.,data=Sensors[t.idsl,],laplace = 1)
Model7
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##      Cold       Hot     Toxic
```

```
## 0.3333333 0.3333333 0.3333333
##
## Conditional probabilities:
##        Humidity
## Y            [,1]      [,2]
##   Cold  50.57143 2.5354628
##   Hot   35.37143 6.7217195
##   Toxic 20.71429 0.4583492
##
##        Temperature
## Y            [,1]      [,2]
##   Cold  24.42857 1.4799784
##   Hot   39.97429 4.4406109
##   Toxic 44.59429 0.6756012
##
##        PPM
## Y             [,1]       [,2]
##   Cold   15.30629   2.631430
##   Hot    19.63800   1.684369
##   Toxic 445.44086 103.336888
```

```
# Predict Model NaiveBayes

Predict7 <- predict(Model7, Sensors[-t.idsl,])
Predict7 <- predict(Model7,newdata=DataTest,)
prediction2<-DataTest$Predict7 <- c(Predict7)
confusionMatrix(prediction2,DataTest$Predict7)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Cold Hot Toxic
##      Cold    26   0     0
##      Hot      0  61     0
##      Toxic    0   0     7
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9615, 1)
##     No Information Rate : 0.6489
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Cold Class: Hot Class: Toxic
## Sensitivity               1.0000     1.0000      1.00000
## Specificity               1.0000     1.0000      1.00000
## Pos Pred Value            1.0000     1.0000      1.00000
## Neg Pred Value            1.0000     1.0000      1.00000
## Prevalence                0.2766     0.6489      0.07447
```

```
## Detection Rate            0.2766     0.6489      0.07447
## Detection Prevalence      0.2766     0.6489      0.07447
## Balanced Accuracy         1.0000     1.0000      1.00000
```

Conclusions :

1. KNN without preprocessing:

- KNN is a simple and intuitive algorithm that can be used for classification tasks.
- It doesn't require preprocessing, but it can be sensitive to the scale and distribution of the features.
- It can struggle with high-dimensional data or datasets with irrelevant features.
- KNN's performance heavily depends on choosing an appropriate value for K.

2. KNN with preprocessing:

- Preprocessing techniques like scaling and dimensionality reduction can improve the performance of KNN.
- Scaling ensures that all features contribute equally to the distance calculation.
- Dimensionality reduction techniques can help reduce noise and improve computational efficiency.

3. KNN Grid:

- KNN Grid allows for an automated selection of the optimal value of K.
- It involves evaluating multiple KNN models with different values of K and selecting the best performing model.
- Grid search can be computationally expensive, especially for large datasets.

4. Logistic Regression:

- Logistic Regression is a powerful algorithm for binary classification tasks.
- It assumes a linear relationship between the features and the log-odds of the target variable.
- It can handle large datasets efficiently and provides interpretable results.
- Logistic Regression performs well when the relationship between features and the target variable is roughly linear.

5. Decision Tree:

- Decision Trees are interpretable and can handle both classification and regression tasks.
- They can capture non-linear relationships between features and the target variable.
- Decision Trees are prone to overfitting, especially when the tree becomes too deep.
- Ensemble methods like Random Forest can address this issue and improve the performance.

6. Random Forest:

- Random Forest is an ensemble method that combines multiple decision trees.
- It reduces overfitting by aggregating the predictions of multiple trees.
- Random Forest provides feature importance measures, which can be helpful for feature selection.
- It can handle high-dimensional data and is generally robust to outliers and missing values.

7. Naive Bayes:

- Naive Bayes is a fast and simple algorithm that performs well on large datasets.
- It assumes feature independence, which may not hold in all cases.
- Despite the independence assumption, Naive Bayes can still provide competitive results.
- It is particularly suitable for text classification tasks and works well with high-dimensional data.