UntitledAssignment 3, “Supervised learning final project”

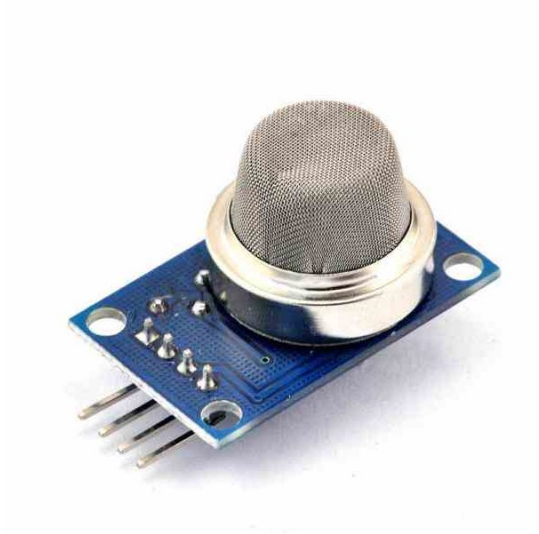
Alejandro Pachón, Santiago Meza, Alexander Morgan

2023-05-29

Sensores

MQ2:

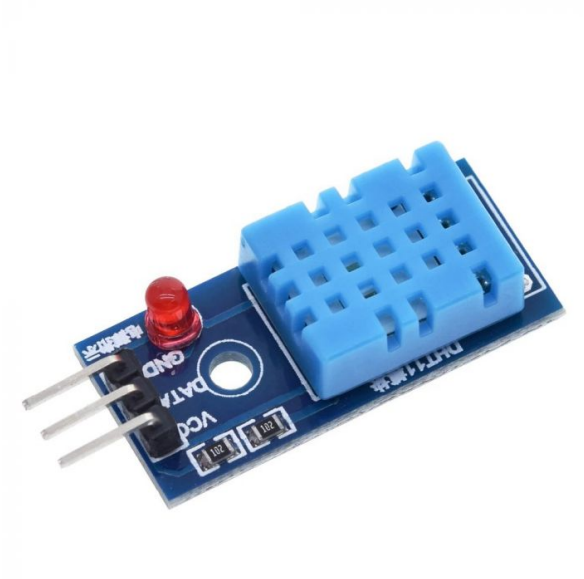
El funcionamiento del sensor MQ2 se basa principalmente en medir la variación en la conductividad de la alterada por la presencia de gases en el aire, estas variaciones producen señales eléctricas que se interpretan de diferentes maneras para llegar al valor de la concentración de gases en el ambiente. Para información más detallada consultar el datasheet.



Para la recolección de datos usando el sensor MQ2, se simuló el ambiente de una cueva volcánica, la cual constaba con irregularidades en el suelo como si fueran grietas, en estas también se agrego alcohol atomizado, se humedecieron partes del ambiente con gasolina y se agrego un poco de gas de un encendedor para simular los gases tóxicos que dichas cuevas pueden llegar a expulsar.

DHT11:

El sensor DHT11 es un sensor capacitivo que mide la humedad y temperatura relativa del aire, incluye también un termistor interno el cual mide la temperatura del ambiente, y muestra los datos mediante una señal digital en el pin de datos, Para información más detallada revisar el datasheet.



En cuanto a la recolección de datos, se tuvieron en cuenta dos de los ambientes simulados, la jungla y el desierto, ya que estos ambientes cambian abruptamente las magnitudes de temperatura y calor debido a la relación entre estas, ya que en el ambiente de jungla se buscaba encontrar una alta humedad y mantener una temperatura relativa estable para que le sensor midiera la relación normalizada en este ambiente, en el ambiente de desierto se busco lo contrario, y se evidencio en el dataset como al elevar la temperatura el factor de humedad disminuye considerable y constantemente.

Hc Sr04

El funcionamiento básico del sensor Hc Sr04 (sensor de ultrasonido) es de la emisión y recepción de pulsos ultrasónicos por sus transductores y medir el tiempo que este tarda en llegar al receptor del sensor. Para información más detallada puede consultar el datasheet.

El sensor de ultrasonido resultó útil en los 3 ambientes, ya que cada uno de estos simulo una distancia de espacio diferente, en el caso de la cueva, al ser ambiente con menor espacio, comparando sus medidas con los demás ambientes son considerables, en el ambiente de jungla al tener diferentes obstáculos y al estar en diferentes alturas, se buscaba simular un espacio con árboles, hojas y/o arbustos, los cuales dificultaran la medición de un espacio exterior, y finalmente en el espacio de desierto se aplicó lo contrario a los dos anteriores escenarios, ya que als er un desierto el espacio de este es muy amplio y esto se refleja en las medidas del dataset.

Modelos de Predicción

Para realizar la partición del dataset, se realizan las líneas de código:

las cuales dan a entender que el dataset tomará una partición de 70 / 30, el cual el 70% será dedicado a la función de entrenar nuestro algoritmo para que sea capaz de realizar predicciones, y el 30% será para realizar las pruebas de estas mismas.

Para asignar las funciones de entrenamiento y pruebas con los datos se realizan las líneas de las funciones respectivas.

```
library(tidyverse)
library(psych)
library(caret)
library(ggplot2)
library(rpart)
library(randomForest)
library(class)
library(gmodels)
DataSet <- DATASETMa
predictors <- colnames(DataSet)[-7]
data.samples <- sample(1:nrow(DataSet),
                       nrow(DataSet) *0.7, replace = FALSE)
sample(DataSet)
training.data <- DataSet[data.samples,c(predictors,"AMBIENTE") ]
```

```
test.data <- DataSet[-data.samples,c(predictors,"AMBIENTE") ]
```

De esta manera ya logramos que en `training.data` quede nuestro nuevo dataset de entrenamiento y en `test.data` nuestros datos de prueba.

Modelo Lineal

Para nuestro modelo de predicción lineal asignamos las variables y la variable de clase al tipo de modelo lineal que vamos a elaborar, en este caso asignamos nuestras variables de temperatura, humedad, distancia, concentración de gases, y la variable predecir será nuestra variables clase la cual será el ambiente en el que se debería encontrar la plataforma robótica, esto se realiza mediante el siguiente código:

```
Modelo Lineal
modelo_multilineal <- lm(DataSet$AMBIENTE ~ DataSet$TEMP + DataSet$HUMEDAD + DataSet$DIS_UP +
                           DataSet$PPM, data = training.data)
summary(modelo_multilineal)
prediccionlm <- predict(modelo_multilineal,test.data)
prediccionlm
RMS1 = data.frame(prediccion = prediccionlm
                         ,actual =DataSet
                         ,RSE = sqrt((prediccionlm-DataSet$AMBIENTE)^2)
)
View(RMS1)
plot(RMS1)
```

Al final de esto, se asigna a `training.data`. Una vez realizado el modelo lineal, se utiliza la función `summary()`para hacer un resumen de estadistica de `modelo_multilineal` y después se puede realizar la predicción de `test.data` y esta se guarda en `prediccionlm`, luego se crea un nuevo dataframe llamado `RMSEmodelo1`, con las columnas de `prediccion` que guarda las predicciones realizadas, `actual` que almacena los valores de `AMBIENTE` y RSE que significa el error estándar residual, este se calcula con la raíz cuadrada de la diferencia al cuadrado entre las predicciones y los valores reales. en este caso sería `sqrt((prediccionlm-DataSet1$AMBIENTE)^2)`.

Árbol de decisión

Este código ajusta un modelo de Random Forest utilizando la función randomForest() . especifica que `AMBIENTE` es la variable objetivo y `PPM`, `TEMP`, `HUMEDAD`y `DIS_UP` son las variables predictoras utilizadas para realizar las predicciones. El modelo se ajusta usando los datos de `training.data`. Utilizando la función `predict()`, se realiza la predicción del conjunto de datos de prueba `test.data` y utilizando el modelo de Random Forest ajustado con `fit.rf`, las predicciones se almacenan en`prediction.rf`. Se crea `output` que combina los datos reales de `test.data$Mileage` con las predicciones del modelo `prediction.rf`. Luego, se calcula el RMSE utilizando la fórmula `sqrt(sum((output$test.data.Mileage - output$prediction.rf)^2)/nrow(output))`, esto calcula la raíz cuadrada de la suma de los errores al cuadrado dividido por el número de observaciones en `output`.

Se crea un marco de datos `RMSEmodelo2` que contiene una columna de `prediccion` con las predicciones del `prediction. If`,Se crea un marco de datos `RMSEmodelo2` que contiene una columna de `prediccion` con las predicciones de `prediction. If`, la columna `ahora` con los valores del conjunto de datos de prueba `test.data$Mileage` y la columna `RSE` con los errores cuadráticos (RSE) calculados utilizando la fórmula `sqrt((prediction.If- test.data$mileage)^2)`

```
# Arbol de decision
fit.rf <- randomForest(AMBIENTE ~ PPM + TEMP + HUMEDAD + DIS_UP, data = training.data)


prediction.rf <- predict(fit.rf, test.data)
```

```
output <- data.frame(test.data$Mileage, prediction.rf)
RMSE = sqrt(sum((output$test.data.Mileage - output$prediction.rf)^2)/
              nrow(output))

RMSE
RMSEmodelo2 = data.frame(prediccion = prediction.rf
                         ,ahora = test.data$Mileage
                         ,RSE = sqrt((prediction.rf-test.data$mileage)^2)
)
```

Modelo GLM

Al igual que en los modelos anteriores se asignan nuestras variables teniendo en cuenta que la variable a predecir es la variable clase AMBIENTE, se declara la función summary al Modelo2 y después de esto se realiza la predicción con los datos de prueba, para finalizar se agrega RMS2 que contiene tres columnas: "prediccion", que almacena las predicciones de "ahora", que contiene los valores de la variable "AMBIENTE" en `test.data`, y "RSE" que representa el error estándar residual calculado como la raíz cuadrada de la diferencia al cuadrado entre las predicciones y los valores. Finalmente tenemos el `View(RMS2)` para ver los resultados de la prediccion.

```
Modelo2 <- glm(AMBIENTE ~ TEMP + HUMEDAD + DIS_UP +
                 PPM, data = training.data)
summary(Modelo2)
modeloPredictivo2 <- predict(Modelo2,test.data)
RMS2 = data.frame(prediccion = modeloPredictivo2
                  ,ahora =test.data$AMBIENTE
                  ,RSE = sqrt((modeloPredictivo2-test.data$AMBIENTE)^2)
)
View(RMS2)
```

Modelo knn

Primero convertimos la variable `AMBIENTE` a un tipo factor con la función `as.factor`, esto para que se trate como una variable no numérica si no una variable de categoría. en codigo sería algo como: `DataSet$AMBIENTE <- as.factor(DataSet$AMBIENTE)`. seguido de esto se genera una proporción 70/30 de `DataSet` y se almacena en `data.samples`, despues se definen las variables predictoras, se crea el data.samples que es el conjunto de datos de entrenamiento, también se crea el conjunto de datos de prueba, y se define el control y el tipo de entrenamiento usando la función `trainControl()`, al ser un modelo de k-vecinos, se utiliza la función `train()` y se ajusta para el modelo usando los predictores, se especifica el método knn y se aplica un centrado y escalado de las variables predictivas, estas se evalúan en 20 valores diferentes de k.Para finalizar muestra el resultado del ajuste con `knnFit` y `plot(knnFit)`.

Para realizar la predicción de los datos, se usa la función `predict()` y se le aplica el conjunto de datos de prueba, esta predicción que guardada en `knnPredict` y para finalizar la predicción se calcula la matriz de confusión usando la funcion `confucionMatrix()` y se comparan las predicciones de `knnPredict` y `test.data$AMBIENTE`

```
DataSet$AMBIENTE <- as.factor(DataSet$AMBIENTE)
data.samples<- sample(1:nrow(DataSet)
                      , nrow(DataSet)*0.7
                      , replace = F)

predictors <- c("TEMP","HUMEDAD","DIS_UP","PPM")

training.data <-
  DataSet[data.samples,c(predictors,"AMBIENTE"),drop=F]
```

```
test.data <-
  DataSet[-data.samples,c(predictors,"AMBIENTE"),drop=F]

ctrl <- trainControl(method = "cv",p=7)
knnFit <- train(AMBIENTE ~ TEMP + HUMEDAD + DIS_UP + PPM
                , data = training.data
                , method = "knn",trControl=ctrl
                , preProcess = c("center","scale")
                , tuneLength = 20)


knnPredict <- predict(knnFit,newdata = test.data)
caret::confusionMatrix(knnPredict,test.data$AMBIENTE)
knnFit
plot(knnFit)
```