Deliverable_2

Julian Abello Orozco

2023-10-08

KNN, Linear regression, and multilinear regression, In a diabetes_012 Dataset

Part 1: Data exploration and data wrangling

In this R Markdown document i will use a data set containing 22 variables that contains 253680 objects. With this dataset I will show how to apply data analysis, Knn, linear and miltilinear regression.

In order to start, it is necessary to load the data set into the program as shown in the following section of the code.

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

parentFolder <- dirname (folder)
data_set_dia <-
  read.csv(paste0(parentFolder,"/dataset/diabetes_012_health_indicators_BRFSS2015.csv"))
```

After loading our data set we must inspect and analyze the information contained in this file. In the following image we can see the variables and brief information about their content.

Figure 1: Characteristics of the data set variables

later using the function `psych` we can extract a statistical analysis of the 22 variables contained in the dataset, which include the mean, standard deviation, minimum and maximum range, among others.

Finally, using the `mutate` function we are going to transform all the data that are not "= 0" in the variable Diabetes_012, then we will show in a small table how many data were classified as "0" or "1"" in this variable

of our set of data

```
test_diabetes<- data_set_dia %>% mutate(Diabetes_012 = ifelse(Diabetes_012!= "0", "1",Diabetes_012))
```

```
Conteo_Diabetes
```

```
##
##      0      1
## 213703  39977
```

Part 2: KNN

KNN DIABETES PREDICTION

First Prediction

In this part of the document we will use the KNN predictive method, for this we will use 3 different variables to achieve the predictions. First, through a stratified sample, we will take approximately 1% of the data to train our models.

```
ss_diabetes <- test_diabetes %>%
  group_by(Diabetes_012) %>%
  sample_n(1269, replace = TRUE) %>%
  ungroup()
```

```
Conteo_ss_Diabetes
```

```
##
##    0    1
## 1269 1269
```

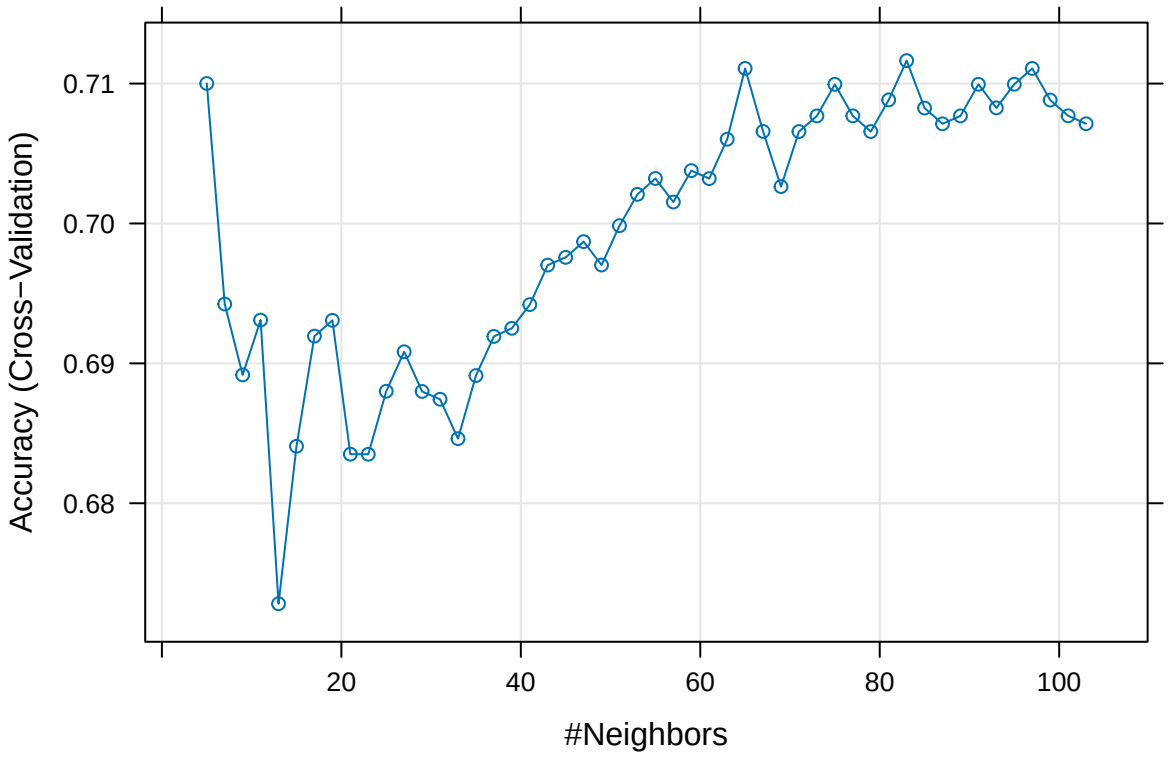
At this point we will find the appropriate number of "K" and we will train the Knn model to predict Diabetes

```
set.seed(123)
ss_diabetes_knn <- ss_diabetes %>%
  group_by(Diabetes_012) %>%
  sample_n(1269, replace = TRUE) %>%
  ungroup()


sample.index <- sample(1:nrow(ss_diabetes_knn)
                       ,nrow(ss_diabetes_knn)*0.7
                       ,replace = F)


predictors <- c("HighBP", "HighChol", "CholCheck", "BMI", "Smoker", "Stroke", "HeartDiseaseorAttack", "

train.data <- ss_diabetes_knn[sample.index, c(predictors, "Diabetes_012"), drop = FALSE]
test.data <- ss_diabetes_knn[-sample.index, c(predictors, "Diabetes_012"), drop = FALSE]


train.data$Diabetes_012 <- factor(train.data$Diabetes_012)
test.data$Diabetes_012 <- factor(test.data$Diabetes_012)
```

These lines of code are related to training and evaluating a machine learning model using the caret library in R, specifically a k-Nearest Neighbors (KNN) model.

```
ctrl <- trainControl(method = "cv", p = 0.7)
knnFit <- train(Diabetes_012 ~ .
```

```
                , data = train.data
                , method = "knn", trControl = ctrl
                , preProcess = c("range") # c("center", "scale") for z-score
                , tuneLength = 50)

plot(knnFit)
```



This code in R performs prediction of a pre-trained K-Nearest Neighbors (KNN) model on new data (test set) and creates a confusion matrix to evaluate the performance of the model on the test data.

```
# Make predictions
knnPredict <- predict(knnFit, newdata = test.data)

# Creates the confusion matrix
confusionMatrix(data = knnPredict, reference = test.data$Diabetes_012)
```

This part of the code uses the `confusionMatrix()` function to calculate a confusion matrix and various metrics to evaluate the performance of the KNN model on the predictions made on the test set.

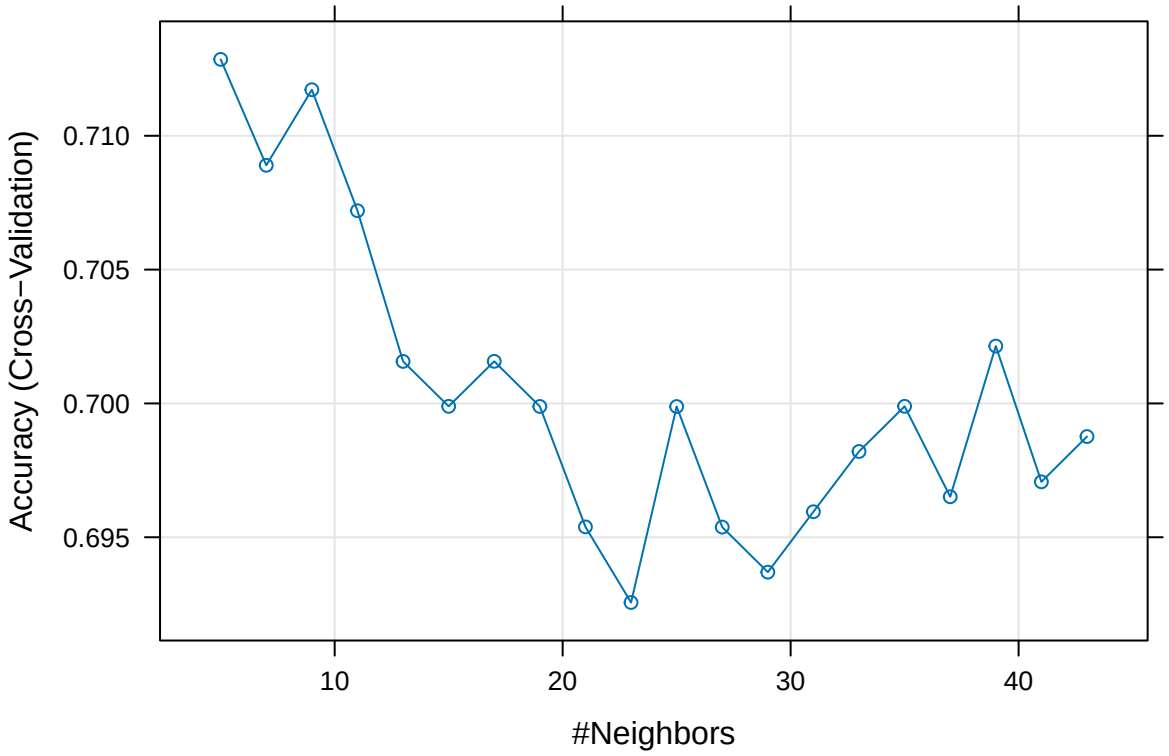
```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction   0   1
##          0 248 103
##          1 130 281
##
##                Accuracy : 0.6942
```

```
##                  95% CI : (0.6602, 0.7268)
##     No Information Rate : 0.5039
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.3881
##
##  Mcnemar's Test P-Value : 0.08851
##
##             Sensitivity : 0.6561
##             Specificity : 0.7318
##          Pos Pred Value : 0.7066
##          Neg Pred Value : 0.6837
##              Prevalence : 0.4961
##          Detection Rate : 0.3255
##    Detection Prevalence : 0.4606
##       Balanced Accuracy : 0.6939
##
##        'Positive' Class : 0
##
```

Second Prediction

In this process, a vector called **predictors_to_remove** is created to specify the variables for removal from the dataset. Subsequently, a new dataset named **train.data2** is generated by excluding the columns listed in **predictors_to_remove** from the original dataset **train.data**. This exclusion is achieved using the **%in%** operator. The training control is configured, employing 5-fold cross-validation (CV) for evaluating the model's performance. Finally, a graphical representation is produced to visualize the K hyperparameter tuning process applied to the trained K-Nearest Neighbors (KNN) model. The x-axis displays the tested K values, while the y-axis illustrates a performance metric (e.g., precision, root mean square error, etc.) for each K value.

```r
predictors_to_remove <- c("NoDocbcCost", "PhysHlth", "DiffWalk", "Education", "Income")
train.data2 <- train.data[, !(names(train.data) %in% predictors_to_remove)]
test.data2 <- test.data[, !(names(test.data) %in% predictors_to_remove)]


ctrl <- trainControl(method = "cv", number = 5)
knnFit2 <- train(Diabetes_012 ~ .
                 , data = train.data2
                 , method = "knn", trControl = ctrl
                 , preProcess = c("range") # c("center", "scale") for z-score
                 , tuneLength = 20)

plot(knnFit2)
```




```
# Make predictions
knnPredict2 <- predict(knnFit2, newdata = test.data2)

# Creates the confusion matrix
confusionMatrix(data = knnPredict2, reference = test.data2$Diabetes_012)
```

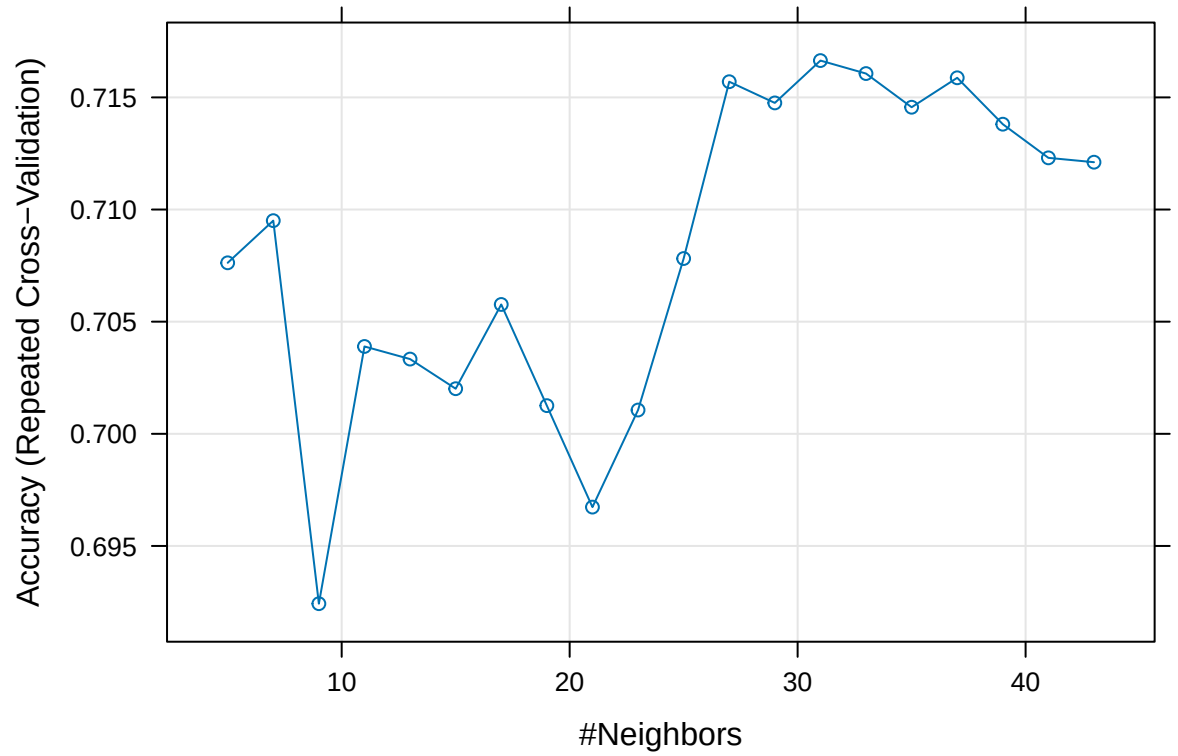
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 253 101
##          1 125 283
##
##                Accuracy : 0.7034
##                  95% CI : (0.6696, 0.7357)
##     No Information Rate : 0.5039
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4065
##
##  Mcnemar's Test P-Value : 0.126
##
##             Sensitivity : 0.6693
##             Specificity : 0.7370
##          Pos Pred Value : 0.7147
##          Neg Pred Value : 0.6936
```

```
##              Prevalence : 0.4961
##          Detection Rate : 0.3320
##    Detection Prevalence : 0.4646
##       Balanced Accuracy : 0.7031
##
##        'Positive' Class : 0
##
```

Third Prediction

Finally, the process is carried out again but this time only using K. Testing the performance of the model by using 3 repeated 10-fold cross-validations.

```
predictors_to_remove2 <- c("Veggies", "MentHlth","HvyAlcoholConsump", "Fruits", "ChoclCheck")
train.data3 <- train.data2[, !(names(train.data2) %in% predictors_to_remove2)]
test.data3 <- test.data2[, !(names(test.data2) %in% predictors_to_remove2)]

ctrl2 <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
knnFit3 <- train(Diabetes_012 ~ .
                 , data = train.data3
                 , method = "knn", trControl = ctrl2
                 , preProcess = c("range") # c("center", "scale") for z-score
                 , tuneLength = 20)

plot(knnFit3)
```



```r
knnPredict3 <- predict(knnFit3, newdata = test.data3)

# Creates the confusion matrix
confusionMatrix(data = knnPredict3, reference = test.data3$Diabetes_012)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 249  96
##          1 129 288
##
##                Accuracy : 0.7047
##                  95% CI : (0.6709, 0.7369)
##     No Information Rate : 0.5039
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.409
##
##  Mcnemar's Test P-Value : 0.0329
##
##             Sensitivity : 0.6587
##             Specificity : 0.7500
##          Pos Pred Value : 0.7217
##          Neg Pred Value : 0.6906
##              Prevalence : 0.4961
##          Detection Rate : 0.3268
##    Detection Prevalence : 0.4528
##       Balanced Accuracy : 0.7044
##
##        'Positive' Class : 0
##
```

The K-Nearest Neighbors (KNN) predictive method is applied to three different sets of variables for diabetes prediction. Firstly, a stratified sample comprising approximately 1% of the data is created for model training. The appropriate value of "K" is determined, and a KNN model is trained to predict diabetes. The model's performance is evaluated using cross-validation with a graphical representation of the K hyperparameter tuning process. In the second prediction, a vector of variables to remove is defined, resulting in a modified dataset used to train a KNN model, followed by model evaluation. Lastly, the process is repeated using different variables and a repeated 10-fold cross-validation for model assessment. Confusion matrices are generated to evaluate the performance of each model.

KNN HeartDiseaseorAttack Prediction

Subsequently, each step is performed again but this time to predict the HeartDiseaseorAttack variable.

First Prediction

```r
set.seed(123)
ss_heartDiseaseorAttack <- ss_diabetes %>%
  group_by(HeartDiseaseorAttack) %>%
  sample_n(1269, replace = TRUE) %>%
  ungroup()
```

```r
predictors <- c("Diabetes_012","HighBP", "HighChol", "CholCheck", "BMI", "Smoker", "Stroke",  "PhysActi

# Original data
train.data <- ss_heartDiseaseorAttack[sample.index, c(predictors, "HeartDiseaseorAttack"), drop = FALSE]
test.data <- ss_heartDiseaseorAttack[-sample.index, c(predictors, "HeartDiseaseorAttack"), drop = FALSE]

train.data$HeartDiseaseorAttack <- factor(train.data$HeartDiseaseorAttack)
test.data$HeartDiseaseorAttack <- factor(test.data$HeartDiseaseorAttack)

# Train the k-NN model
ctrl <- trainControl(method = "cv", p = 0.7)
knnFit <- train(HeartDiseaseorAttack ~ .
                , data = train.data
                , method = "knn", trControl = ctrl
                , preProcess = c("range") # c("center", "scale") for z-score
                , tuneLength = 50)

# Make predictions
knnPredict <- predict(knnFit, newdata = test.data)

# Creates the confusion matrix
# Original data
train.data <- ss_heartDiseaseorAttack[sample.index, c(predictors, "HeartDiseaseorAttack"), drop = FALSE]
test.data <- ss_heartDiseaseorAttack[-sample.index, c(predictors, "HeartDiseaseorAttack"), drop = FALSE]

train.data$HeartDiseaseorAttack <- factor(train.data$HeartDiseaseorAttack)
test.data$HeartDiseaseorAttack <- factor(test.data$HeartDiseaseorAttack)

# Train the k-NN model
ctrl <- trainControl(method = "cv", p = 0.7)
knnFit <- train(HeartDiseaseorAttack ~ .
                , data = train.data
                , method = "knn", trControl = ctrl
                , preProcess = c("range") # c("center", "scale") for z-score
                , tuneLength = 50)

# Make predictions
knnPredict <- predict(knnFit, newdata = test.data)

# Creates the confusion matrix
confusionMatrix(data = knnPredict, reference = test.data$HeartDiseaseorAttack)
```

Second Prediction

```r
predictors_to_remove <- c("AnyHealthcare", "NoDocbcCost", "DiffWalk", "Education", "Income")
train.data2 <- train.data[, !(names(train.data) %in% predictors_to_remove)]
test.data2 <- test.data[, !(names(test.data) %in% predictors_to_remove)]

# Train the k-NN model
ctrl <- trainControl(method = "cv", number = 5)
knnFit2 <- train(HeartDiseaseorAttack ~ .
                 , data = train.data2
                 , method = "knn", trControl = ctrl
```

```
                  , preProcess = c("range") # c("center", "scale") for z-score
                  , tuneLength = 50)


# Make predictions
knnPredict2 <- predict(knnFit2, newdata = test.data2)

# Creates the confusion matrix
confusionMatrix(data = knnPredict2, reference = test.data2$HeartDiseaseorAttack)
```

Third Prediction

```
predictors_to_remove2 <- c("ChoclCheck", "MentHlth","HvyAlcoholConsump", "Fruits", "Veggies")
train.data3 <- train.data2[, !(names(train.data2) %in% predictors_to_remove2)]
test.data3 <- test.data2[, !(names(test.data2) %in% predictors_to_remove2)]

# Train the k-NN model
ctrl2 <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
knnFit3 <- train(HeartDiseaseorAttack ~ .
                  , data = train.data3
                  , method = "knn", trControl = ctrl2
                  , preProcess = c("range") # c("center", "scale") for z-score
                  , tuneLength = 50)


# Make predictions
knnPredict3 <- predict(knnFit3, newdata = test.data3)

# Creates the confusion matrix
confusionMatrix(data = knnPredict3, reference = test.data3$HeartDiseaseorAttack)
```

KNN Find Sex Prediction

And finally the knn process is repeated again to predict the sex

First Prediction

```
###KNN Models and Experiments to Find Sex


## selection of 1500 samples of each factor of the dataset#
set.seed(123)
ss_sex <- ss_diabetes %>%
  group_by(Sex) %>%
  sample_n(1269, replace = TRUE) %>%
  ungroup()

predictors <- c("Diabetes_012","HighBP", "HighChol", "CholCheck", "BMI", "Smoker", "Stroke", "HeartDisea

# Original data
train.data <- ss_sex[sample.index, c(predictors, "Sex"), drop = FALSE]
test.data <- ss_sex[-sample.index, c(predictors, "Sex"), drop = FALSE]
```

```r
train.data$Sex <- factor(train.data$Sex)
test.data$Sex <- factor(test.data$Sex)

# Train the k-NN model
ctrl <- trainControl(method = "cv", p = 0.7)
knnFit <- train(Sex ~ .
                , data = train.data
                , method = "knn", trControl = ctrl
                , preProcess = c("range") # c("center", "scale") for z-score
                , tuneLength = 50)


# Make predictions
knnPredict <- predict(knnFit, newdata = test.data)

# Creates the confusion matrix
confusionMatrix(data = knnPredict, reference = test.data$Sex)
```

Second Prediction

```r
predictors_to_remove <- c("AnyHealthcare", "NoDocbcCost", "DiffWalk", "Age", "PhysActivity")
train.data2 <- train.data[, !(names(train.data) %in% predictors_to_remove)]
test.data2 <- test.data[, !(names(test.data) %in% predictors_to_remove)]

# Train the k-NN model
ctrl <- trainControl(method = "cv", number = 5)
knnFit2 <- train(Sex ~ .
                 , data = train.data2
                 , method = "knn", trControl = ctrl
                 , preProcess = c("range") # c("center", "scale") for z-score
                 , tuneLength = 50)

#Make predictions
knnPredict2 <- predict(knnFit2, newdata = test.data2)

# Creates the confusion matrix
  confusionMatrix(data = knnPredict2, reference = test.data2$Sex)
```

Third Prediction

```r
predictors_to_remove2 <- c("ChoclCheck", "MentHlth","HvyAlcoholConsump", "Fruits", "Veggies")
train.data3 <- train.data2[, !(names(train.data2) %in% predictors_to_remove2)]
test.data3 <- test.data2[, !(names(test.data2) %in% predictors_to_remove2)]

# Train the k-NN model
ctrl2 <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
knnFit3 <- train(Sex ~ .
                 , data = train.data3
                 , method = "knn", trControl = ctrl2
                 , preProcess = c("range") # c("center", "scale") for z-score
                 , tuneLength = 50)
```

```r
#Make predictions
knnPredict3 <- predict(knnFit3, newdata = test.data3)

# Creates the confusion matrix
confusionMatrix(data = knnPredict3, reference = test.data3$Sex)
```

Part 3: Linear regression model BM

First Prediction

1. **ins_model <- lm(BMI ~ ., data = train.data)**: Ajusta un modelo de regresión lineal utilizando la función "lm()". La variable "BMI" se utiliza como variable de respuesta y todas las demás variables como predictoras. Luego, se muestra un resumen del modelo con "summary(ins_model)".
2. **train.control <- trainControl(method = "cv", number = 10)**: Configura el control de entrenamiento para realizar una validación cruzada de 10 pliegues.
3. **model <- train(BMI ~ ., data = train.data, method = "lm", trControl = train.control)**: Entrena un modelo de regresión lineal utilizando la función "train()" de la biblioteca "caret". Se utiliza una validación cruzada de 10 pliegues y se muestra un resumen del modelo.

```r
folder <- dirname(rstudioapi :: getSourceEditorContext()$path)

parentFolder <- dirname (folder)
data <-
  read.csv(paste0(parentFolder,"/dataset/diabetes_012_health_indicators_BRFSS2015.csv"))

data$Diabetes_012 <- ifelse(data$Diabetes_012 == 0, 0, 1)

set.seed(1)
data_estratificada2 <- data[sample(nrow(data), 1269), ]

predictors <- colnames(data_estratificada2)[-5]
sample.index <- sample(1:nrow(data_estratificada2),
                       nrow(data_estratificada2) * 0.7,
                       replace = FALSE)


train.data <- data_estratificada2[sample.index, c(predictors, "BMI"), drop = FALSE]
test.data <- data_estratificada2[-sample.index, c(predictors, "BMI"), drop = FALSE]

ins_model <- lm(BMI ~ ., data = train.data)

summary(ins_model)
```

```
##
## Call:
## lm(formula = BMI ~ ., data = train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.820  -3.710  -0.727   2.804  57.956
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          32.538296   2.083364  15.618  < 2e-16 ***
```

```
## Diabetes_012          3.315183   0.618347   5.361 1.06e-07 ***
## HighBP                2.911576   0.483318   6.024 2.51e-09 ***
## HighChol             -0.252376   0.463680  -0.544 0.586382
## CholCheck             1.117145   1.193433   0.936 0.349495
## Smoker               -0.760655   0.446857  -1.702 0.089070 .
## Stroke               -2.090628   1.285079  -1.627 0.104133
## HeartDiseaseorAttack -1.153897   0.789386  -1.462 0.144168
## PhysActivity         -1.905908   0.575568  -3.311 0.000967 ***
## Fruits               -0.794234   0.471595  -1.684 0.092514 .
## Veggies               0.695635   0.575964   1.208 0.227464
## HvyAlcoholConsump    -0.965965   1.037006  -0.931 0.351857
## AnyHealthcare         1.080104   1.085865   0.995 0.320163
## NoDocbcCost          -1.682749   0.844785  -1.992 0.046693 *
## GenHlth               0.262895   0.268332   0.980 0.327489
## MentHlth              0.005084   0.031279   0.163 0.870919
## PhysHlth              0.007917   0.031902   0.248 0.804065
## DiffWalk              1.979468   0.723099   2.737 0.006318 **
## Sex                  -0.461168   0.450371  -1.024 0.306134
## Age                  -0.528892   0.080161  -6.598 7.26e-11 ***
## Education            -0.324908   0.243971  -1.332 0.183293
## Income               -0.046984   0.133374  -0.352 0.724723
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.368 on 866 degrees of freedom
## Multiple R-squared:  0.1809, Adjusted R-squared:  0.1611
## F-statistic:  9.11 on 21 and 866 DF,  p-value: < 2.2e-16
```

```
# Train the model
train.control <- trainControl(method = "cv", number = 10 )
model <- train(BMI ~ ., data = train.data, method = "lm",
               trControl = train.control)

# Summarize the results
print(model)
```

```
## Linear Regression
##
## 888 samples
##  21 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 799, 799, 798, 798, 800, 801, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   6.362476  0.1642812  4.474592
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Second Prediction

```r
predictors_to_remove <- c("AnyHealthcare", "CholCheck", "MentHlth", "Education", "Sex")

train.data2 <- train.data[, !(names(train.data) %in% predictors_to_remove)]
test.data2 <- test.data[, !(names(test.data) %in% predictors_to_remove)]

ins_model <- lm(BMI ~ ., data = train.data2)

summary(ins_model)

# Train the model
train.control <- trainControl(method = "cv", number = 5)
model <- train(BMI ~ ., data = train.data2, method = "lm",
               trControl = train.control)

# Summarize the results
print(model)
```

Third Prediction

```r
predictors_to_remove <- c("Income", "Stroke", "NoDocbcCost", "Veggies", "HvyAlcoholConsump")

train.data3 <- train.data2[, !(names(train.data2) %in% predictors_to_remove)]
test.data3 <- test.data2[, !(names(test.data2) %in% predictors_to_remove)]

ins_model <- lm(BMI ~ ., data = train.data3)

summary(ins_model)

# Train the model
train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
model <- train(BMI ~ ., data = train.data3, method = "lm",
               trControl = train.control)
# Summarize the results
print(model)
```

Linear regression model MentHlth

This code section refers to a linear regression analysis applied to the response variable “MentHlth” using the dataset “data_estratificada2.

First Prediction

```r
### Linear regression model MentHlth

set.seed(123)
data_estratificada2 <- data[sample(nrow(data), 1269), ]

predictors <- colnames(data_estratificada2)[-16]
sample.index <- sample(1:nrow(data_estratificada2),
                       nrow(data_estratificada2) * 0.7,
                       replace = FALSE)

### ENTRENAMIENTO
```

```r
train.data <- data_estratificada2[sample.index, c(predictors, "MentHlth"), drop = FALSE]
test.data <- data_estratificada2[-sample.index, c(predictors, "MentHlth"), drop = FALSE]

ins_model <- lm(MentHlth ~ ., data = train.data)
summary(ins_model)

# Train the model
train.control <- trainControl(method = "cv", number = 10 )
model <- train(MentHlth ~ ., data = train.data, method = "lm",
               trControl = train.control)

# Summarize the results
print(model)
```

Second Prediction

```r
predictors_to_remove <- c("BMI", "HeartDiseaseorAttack", "Stroke", "PhysActivity", "CholCheck")

train.data2 <- train.data[, !(names(train.data) %in% predictors_to_remove)]
test.data2 <- test.data[, !(names(test.data) %in% predictors_to_remove)]

ins_model <- lm(MentHlth ~ ., data = train.data2)
summary(ins_model)

# Train the model
train.control <- trainControl(method = "cv", number = 5)
model <- train(MentHlth ~ ., data = train.data2, method = "lm",
               trControl = train.control)

# Summarize the results
print(model)
```

Third Prediction

```r
predictors_to_remove <- c("Diabetes_012", "HighBP", "HighChol", "Veggies", "Education")

train.data3 <- train.data2[, !(names(train.data2) %in% predictors_to_remove)]
test.data3 <- test.data2[, !(names(test.data2) %in% predictors_to_remove)]

ins_model <- lm(MentHlth ~ ., data = train.data3)
summary(ins_model)

# Train the model
train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
model <- train(MentHlth ~ ., data = train.data3, method = "lm",
               trControl = train.control)
# Summarize the results
print(model)
```

Linear regression model PhysHlth

First Prediction

```
#### Linear regression model PhysHlth
set.seed(123)
data_estratificada3 <- data[sample(nrow(data), 1269), ]

predictors <- colnames(data_estratificada2)[-17]
sample.index <- sample(1:nrow(data_estratificada3),
                       nrow(data_estratificada3) * 0.7,
                       replace = FALSE)

train.data <- data_estratificada2[sample.index, c(predictors, "PhysHlth"), drop = FALSE]
test.data <- data_estratificada2[-sample.index, c(predictors, "PhysHlth"), drop = FALSE]

ins_model <- lm(PhysHlth ~ ., data = train.data)
summary(ins_model)

# Train the model
train.control <- trainControl(method = "cv", number = 10 )
model <- train(PhysHlth ~ ., data = train.data, method = "lm",
               trControl = train.control)
# Summarize the results
print(model)
```

Second Prediction

```
predictors_to_remove <- c("Sex", "DiffWalk", "Diabetes_012", "CholCheck", "Income")

train.data2 <- train.data[, !(names(train.data) %in% predictors_to_remove)]
test.data2 <- test.data[, !(names(test.data) %in% predictors_to_remove)]

ins_model <- lm(PhysHlth ~ ., data = train.data2)
summary(ins_model)

# Train the model
train.control <- trainControl(method = "cv", number = 5)
model <- train(PhysHlth ~ ., data = train.data2, method = "lm",
               trControl = train.control)
# Summarize the results
print(model)
```

Third Prediction

```
predictors_to_remove <- c("Fruits", "HeartDiseaseorAttack", "BMI", "Veggies", "Fruits")

train.data3 <- train.data2[, !(names(train.data2) %in% predictors_to_remove)]
test.data3 <- test.data2[, !(names(test.data2) %in% predictors_to_remove)]

ins_model <- lm(PhysHlth ~ ., data = train.data3)
summary(ins_model)
```

```r
# Train the model
train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
model <- train(PhysHlth ~ ., data = train.data3, method = "lm",
               trControl = train.control)
# Summarize the results
print(model)
```

The analysis between the advantages and disadvantages of the k-Nearest Neighbors (KNN) algorithm and linear and multiline regression models is essential to understand which of them is more suitable for a given prediction problem. Below is a comparative analysis:

Advantages of KNN: Conceptual simplicity: KNN is a supervised learning algorithm that is easy to understand and implement. It does not require assumptions about the distribution of the data or the relationship between the variables.

Adaptability to nonlinear data: KNN can capture nonlinear relationships between predictor variables and the target variable, making it versatile for complex problems.

Non-parametric: KNN does not assume a specific form for the relationship between the predictor variables and the target variable, making it suitable for a variety of problems.

Intuitive interpretation: KNN predictions are based on nearest neighbors, allowing for intuitive interpretation of the results.

Disadvantages of KNN: Sensitivity to noise: KNN is sensitive to outliers and noise in the data, which can negatively affect its performance.

Computationally expensive: On large data sets, KNN can be computationally expensive due to the calculation of distances between points.

Need to tune hyperparameters: It is necessary to choose the optimal value of "k" (the number of neighbors) and an appropriate distance metric, which may require tuning and validation.

Advantages of Linear and Multilinear Regression: Clear Interpretation: Linear and multilinear regression models provide coefficients that represent the quantitative relationship between the predictor variables and the target variable, making interpretation easier.

Computational efficiency: These models are typically more computationally efficient than KNN on large data sets.

Less sensitive to noise: Linear regression may be less sensitive to outliers and noise in the data compared to KNN.

Disadvantages of Linear and Multilinear Regression: Restrictive assumptions: Linear and multilinear regression models assume a linear relationship between the predictor variables and the target variable, which may not be valid in many cases.

Inability to capture complex nonlinear relationships: These models may not be able to capture complex nonlinear relationships in the data, which may result in poor fit.

Need for validation of assumptions: It is important to validate the assumptions of normality and homoscedasticity so that the results are reliable.

In conclusion, the choice between KNN and linear and multilinear regression models depends on the specific problem and data characteristics. KNN is suitable when no clear assumptions can be made about the relationship between variables, while linear and multilinear regression are useful when a linear relationship is suspected and a clearer interpretation is sought. Selection of the appropriate method should be based on the nature of the problem and the quality of the available data.