CS 199

HW3 - Predicting Crime Rates

April 8, 2014

Sam Laane <laane2@illinois.edu>
José Vicente Ruiz <ruizcep2@illinois.edu>

Contents

1 Removing variables with missing values

1.1 Implementation

```
# Read the data.
data <- read.csv('communitiesH.data', h=T) # Headers = True.

# Remove non-predictive attributes (including state number).
clean_data <- data[, -c(1:5)]

# Remove the attributes with question marks (unknowns) of them.
unkw <- clean_data == '?'
unkw_per_attr <- apply(unkw, 2, sum) # Sum the columns (2).
attr_with_unkw <- which(unkw_per_attr > 0) # Equivalent to find in
    Matlab.
no_unkw_data <- clean_data[, -attr_with_unkw]
```

2 Basics - Linear regression

2.1 Implementation

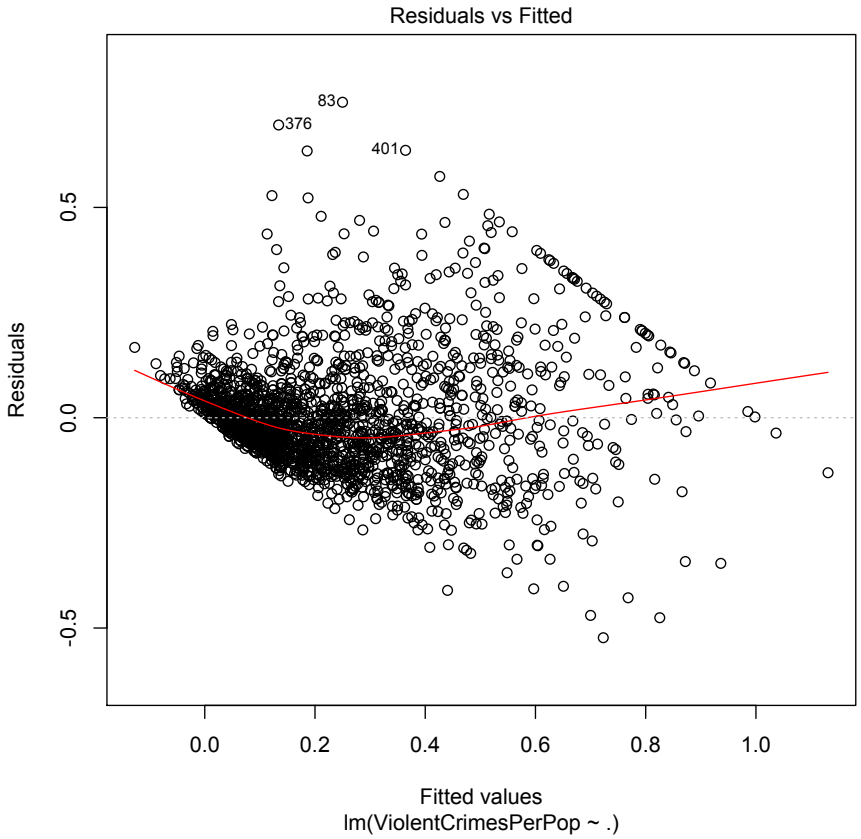
```
# Compute the regression with the data free of unknowns.
linear_regr <- lm(ViolentCrimesPerPop ~ ., data=no_unkw_data)

# Evaluate the regression looking at the mean-squared error on the
    training data.
residuals <- resid(linear_regr)  # Function to get the residuals.
mse_residuals <- sum((residuals - mean(residuals)) ^ 2) / length(
    residuals)
printf(" - Mean-squared error on the whole data: %.2e", mse_residuals)

# Split off some test data and compute a new regression on the rest of
    the data
# (train data).
folds <- cvFolds(nrow(no_unkw_data), K=5)
train_data <- no_unkw_data[folds$subsets[folds$which != 1], ]
test_data  <- no_unkw_data[folds$subsets[folds$which == 1], ]

linear_regr_test <- lm(ViolentCrimesPerPop ~ ., data=train_data)

# Evaluate the regression looking at the mean-squared error on that
    test data.
residuals_test <- test_data$ViolentCrimesPerPop - predict(linear_regr_
    test, test_data)
plot(test_data$ViolentCrimesPerPop, residuals_test)

mse_residuals_test <- sum((residuals_test - mean(residuals_test)) ^ 2)
    / length(residuals_test)
printf(" - Mean-squared error on the test data (20%%): %.2e", mse_
    residuals_test)

# Prepare the data for the Box-Cox tranformation substituting zero
    values by
```

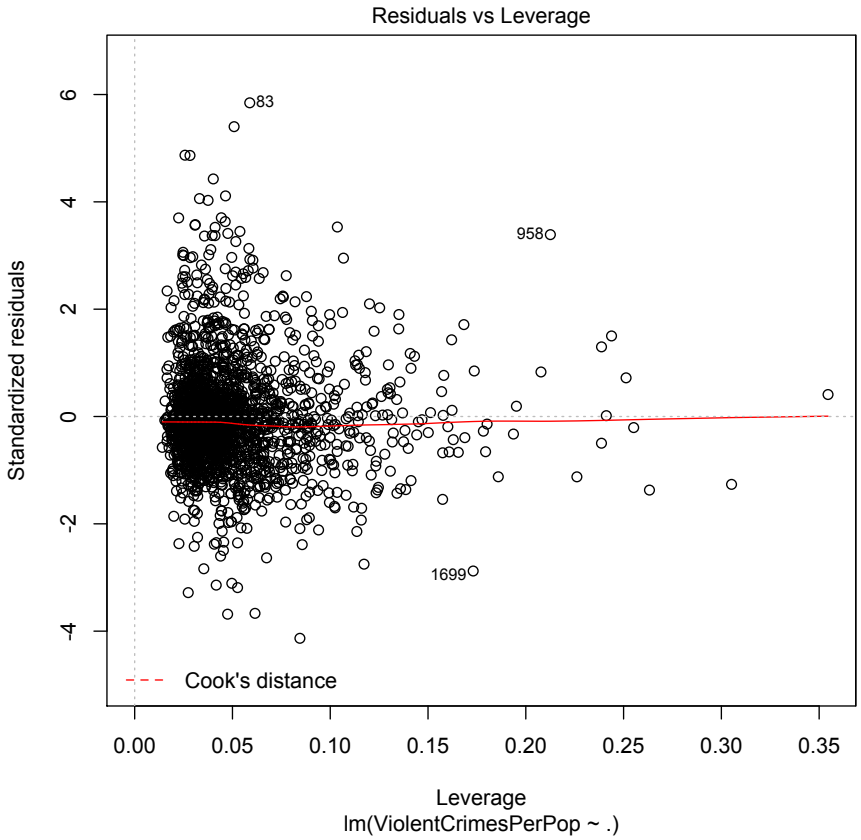
```
25 # a very small number.
26 boxcox_data <- no_unkw_data
27 boxcox_data$ViolentCrimesPerPop[ which(boxcox_data$ViolentCrimesPerPop
      == 0) ] <- 1e-100
28
29 # Apply Box-Cox and get a list of the lamdbas and their log likelihood
      and obtain
30 # the value with the highest likelihood.
31 linear_regr_mod <- lm(ViolentCrimesPerPop ~ ., data=boxcox_data)
32 lambdas <- boxcox(linear_regr_mod, plotit=F)
33 max_lambda <- lambdas$x[ which(lambdas$y == max(lambdas$y)) ]
34
35 # Change the very low values to zero again.
36 boxcox_data$ViolentCrimesPerPop[which(boxcox_data$ViolentCrimesPerPop
      == 1e-100)] <- 0
37
38 # Transform the data with the given value of lambda.
39 if (max_lambda == 0) {
40   boxcox_data$ViolentCrimesPerPop <- log(boxcox_data$
        ViolentCrimesPerPop)
41 } else {
42   boxcox_data$ViolentCrimesPerPop <- (boxcox_data$ViolentCrimesPerPop^
        max_lambda - 1)/max_lambda
43 }
44
45 # Compute the linear regression again and plot it.
46 boxcox_regr <- lm(ViolentCrimesPerPop ~ ., data=boxcox_data)
47 box_residuals_test <- sum((boxcox_regr$residuals - mean(boxcox_regr$
      residuals)) ^ 2) / length(boxcox_regr$residuals)
48 printf(" - Mean-squared error on the Boxcox all data: %.2e", box_
      residuals_test)
49 plot(boxcox_regr)
```

2.2 Results

- Mean-squared error on the whole data: 1.66e-02
- Mean-squared error on the test data (20%): 1.88e-02

2.2.1 Standard data





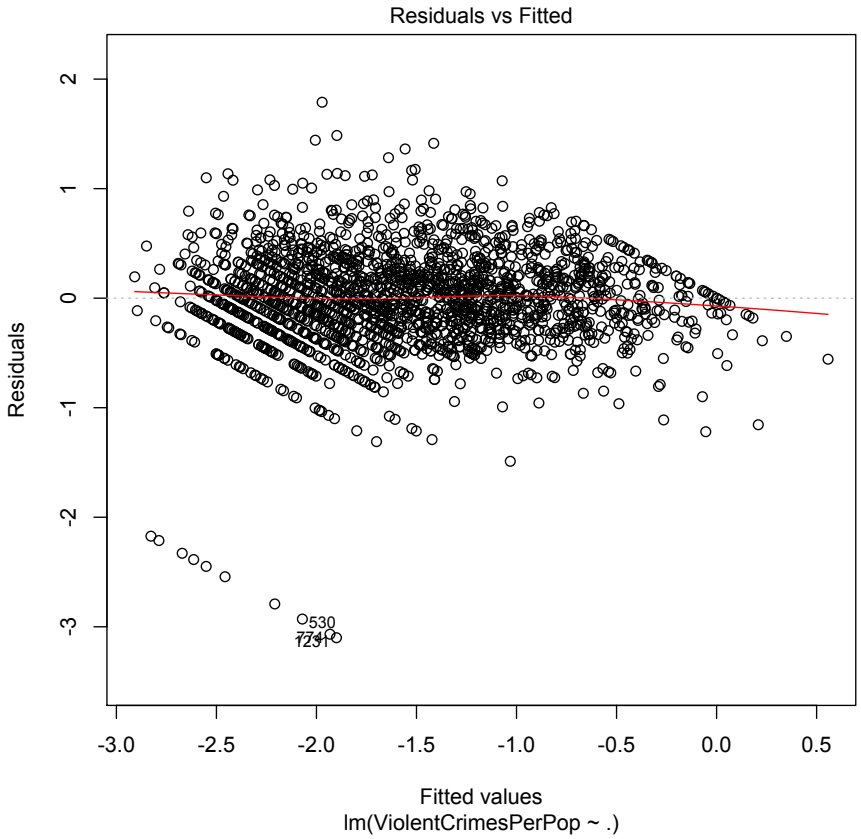
2.3 Conclusions

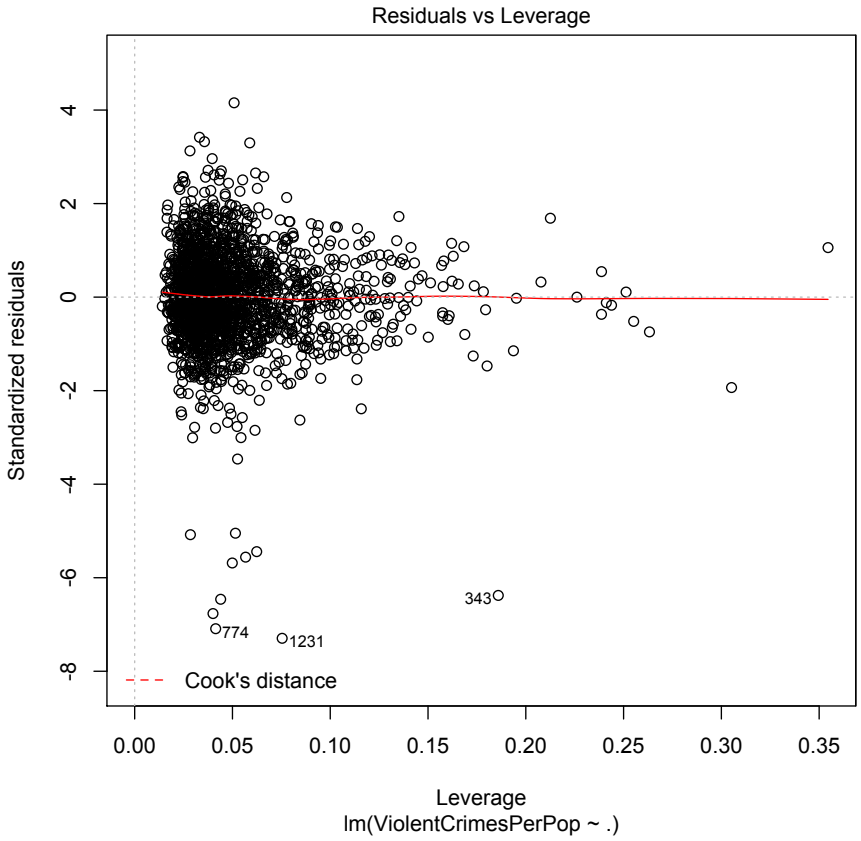
The Mean-squared error on both the full regression and the tested regression are both quite small. The tested regression Mean-squared error is a little larger as it has less data

The residual graph was relatively poorly behaved with some strange linear structures. The lack a “banna” shows that our problem is not likely to be fixed by Boxcox or other types of transformed. We need more explanatory values. According to the Residuals VS Leverage graph no points had undue influence or worry some cooks distance.

This show that the regression is likely possible with our data but needs work.

2.3.1 Box-Cox transformed data





2.4 Results

- Mean-squared error on the Boxcox all data: 1.85e-01

2.5 Conclusions

Our Boxcox transformed data looked promising but has a larger Mean-squared error. The residual graph shows that while the data looks straighter the linear structures can still be seen. However the when one accounts for the change in base in the distortion graph looks far worse. We also see that at low fitted values the data looks split and some have extremely low Residuals. This is likely caused by values extremely close to zero.

Boxcox was useless.

3 Basics - Nearest Neighbor regression

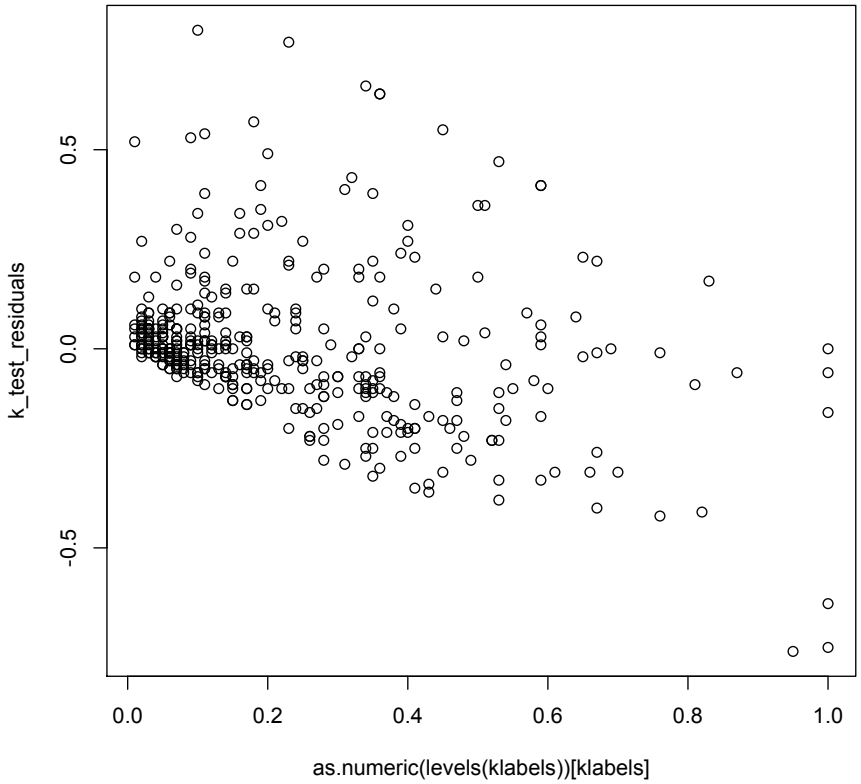
3.1 Implementation

```
############################
printf("Nearest Neighbours:")

# Compute the regression and plot it.
nn_regr <- knn.reg(train_data[!(names(train_data)) %in% c("ViolentCrimesPerPop")],
                   test = NULL,
                   train_data$ViolentCrimesPerPop, k = 1,
                   algorithm = c("kd_tree"))
plot(nn_regr$pred, nn_regr$residuals)
#png("Nearest_Neighbours_all_data.png")

# Evaluate the regression on the above training data with mean-squared error.
mse_residuals_knn <- sum((nn_regr$residuals - mean(nn_regr$residuals)) ^ 2) /
    length(nn_regr$residuals)
printf(" - Mean-squared error on whole data with cross validation: %.2e",
       mse_residuals_knn)

# Compute the regression on the above test data.
klabels <- knn(train_data[!(names(train_data)) %in% c("ViolentCrimesPerPop")],
               test_data [!(names(test_data))  %in% c("ViolentCrimesPerPop")],
               train_data$ViolentCrimesPerPop, k = 1,
               prob = FALSE, algorithm = c("kd_tree"))
k_test_residuals <- test_data$ViolentCrimesPerPop - as.numeric(levels(klabels))[klabels]
plot(as.numeric(levels(klabels))[klabels], k_test_residuals)
# png("Nearest_Neighbours_trainning_data.png")
# Evaluate the previous regression with mean-squared error and print the result.
mse_residuals_knn_test <- sum((k_test_residuals - mean(k_test_residuals)) ^ 2) /
```

```
28      length(k_test_residuals)
29  printf(" - Mean-squared error on the test data (20%%): %.2e", mse_
        residuals_knn_test)
```

3.2 Results

- Mean-squared error on the test data (20%): 3.64e-02



3.3 Conclusions

The Mean-squared error on k-nearest neighbor regression did not work as well as the linear regression. It's Mean-squared error was signifantly higher and looks worse. Scaling the data changed nothing. I am guessing that the data set was prescaled.

4 Dealing with missing values

4.1 Implementation

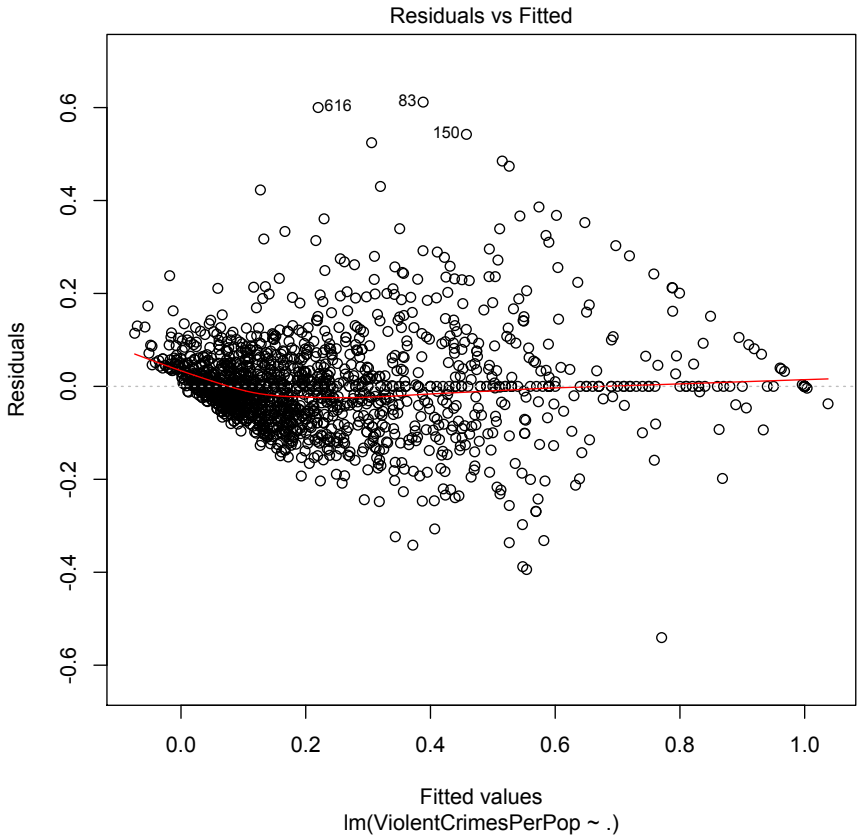
```
1  # Impute unknown values by covering all the columns with unknowns.
2  interpolated_data <- clean_data
3  for(i in attr_with_unkw) {
4      # Divide data by rows in two sets: one with the samples that
           contain a question
5      # mark in that column and one with the rest.
```

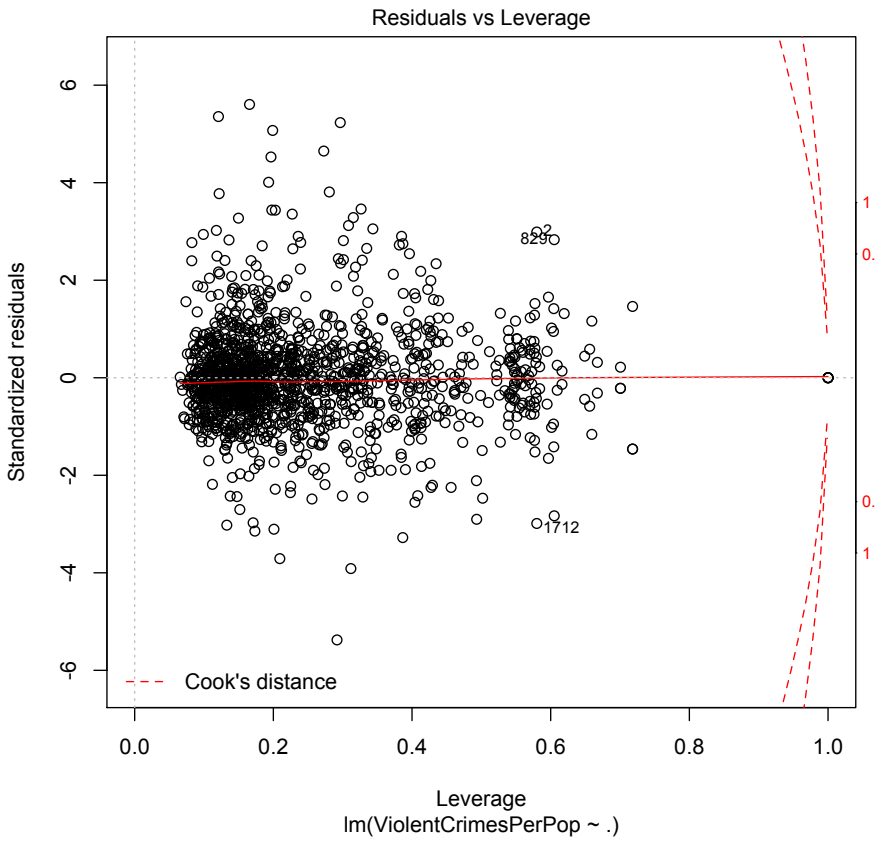
```
6       unk_indices <- which(clean_data[,i] == '?')
7       i_unk_data <- no_unkw_data[unk_indices, ]
8       i_no_unk_data <- no_unkw_data[-unk_indices, ]
9
10      # Compute the nearest neighbours in the set of elements with
            question mark of
11      # the elements with a question mark in that column.
12      i_klabels <- knn(i_no_unk_data,
13                       i_unk_data,
14                       i_no_unk_data$ViolentCrimesPerPop, k = 1,
15                       prob = FALSE, algorithm=c("kd_tree"))
16      indices <- attr(i_klabels, "nn.index")
17      nn_subs <- clean_data[-unk_indices, ][indices,]
18
19      # Substitute the question mark values by the value of the nearest
            neighbour.
20      interpolated_data[unk_indices, i] <- nn_subs[,i]
21  }
22
23  # Split off new test data.
24  folds <- cvFolds(nrow(interpolated_data), K=5)
25  train_data <- interpolated_data[folds$subsets[folds$which != 1], ]
26  test_data  <- interpolated_data[folds$subsets[folds$which == 1], ]
27
28  # Compute a linear regression again and evaluate it with mean-squares.
29  printf("Linear Regression (imputed missing values):")
30  linear_regr_test <- lm(ViolentCrimesPerPop ~ ., data=train_data)
31
32  residuals_test <- test_data$ViolentCrimesPerPop - predict(linear_regr_
        test, test_data)
33  plot(test_data$ViolentCrimesPerPop, residuals_test)
34
35  mse_residuals_test <- sum((residuals_test - mean(residuals_test)) ^ 2)
        / length(residuals_test)
36  printf(" - Mean-squared error on the test data (20%%): %.2e", mse_
        residuals_test)
37
38  # Also, compute a nearest neighbour regression and evaluate it.
39  printf("Nearest Neighbours (imputed missing values):")
40  klabels <- knn(train_data[!(names(train_data)) %in% c("
        ViolentCrimesPerPop")],
41                 test_data [!(names(test_data))  %in% c("
                       ViolentCrimesPerPop")],
42                 train_data$ViolentCrimesPerPop, k = 1,
43                 prob = FALSE, algorithm = c("kd_tree"))
44  k_test_residuals <- test_data$ViolentCrimesPerPop - as.numeric(levels(
        klabels))[klabels]
45  plot(as.numeric(levels(klabels))[klabels], k_test_residuals)
46
47  # Compute the mean-squared error and print the result.
48  mse_residuals_knn_test <- sum((k_test_residuals - mean(k_test_residuals
        )) ^ 2) /
49       length(k_test_residuals)
50  printf(" - Mean-squared error on the test data (20%%): %.2e", mse_
        residuals_knn_test)
```

4.2 Results

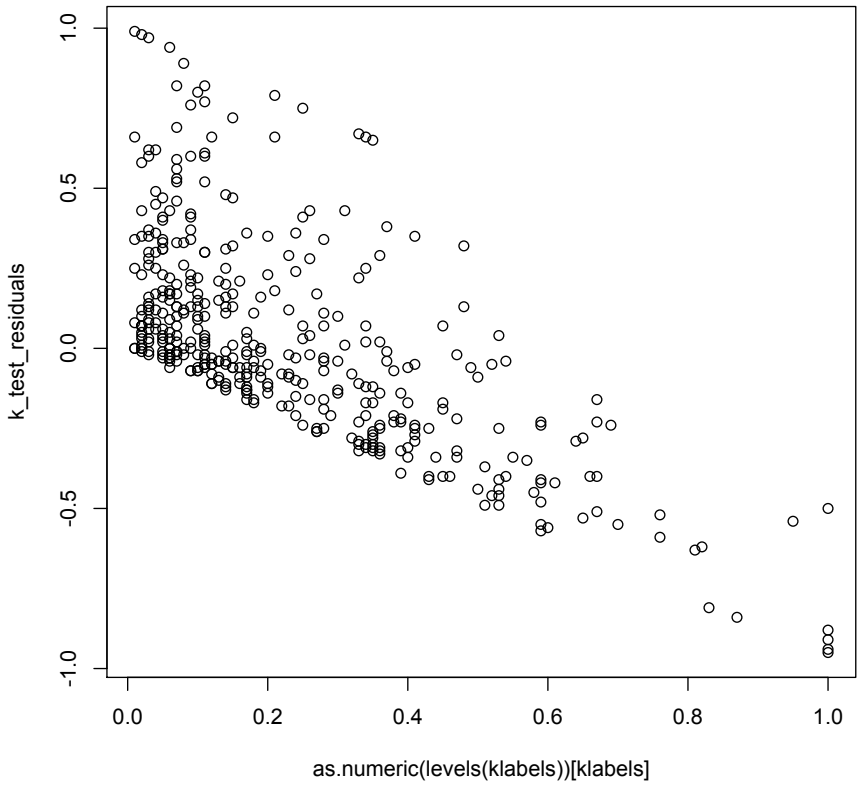
Linear Regression (imputed missing values): - Mean-squared error on the test data (20%): 1.88e-02 Nearest Neighbours (imputed missing values): - Mean-squared error on the test data (20%): 1.00e-01

4.2.1 Linear regression





4.2.2 Nearest Neighbours



4.3 Conclusions

5 Modified Nearest Neighbours

5.1 Implementation

```
# Compute the euclidean distance between two vectors that may contain question
# mark elements in a vectorized way.
distance_vec <- function(v1, v2){
  # Change question mark elements of each vector by the same position elements
  # in the other vector, so that they get cancelled when computing the distance.
  unk_indices_1 <- which(v1 == '?')
  unk_indices_2 <- which(v2 == '?')

  v1[unk_indices_1] = v2[unk_indices_1]
  v2[unk_indices_2] = v1[unk_indices_2]

  # Change question marks elements with same position in both vectors by 0.
  unk_indices <- which(v1 == '?')
  v1[unk_indices] <- 0
```

```
15    v2[unk_indices] <- 0
16
17    # Ensure vectors type is numeric.
18    v1 <- as.numeric(v1)
19    v2 <- as.numeric(v2)
20
21    # Return the euclidian distance.
22    return(sqrt(sum((v1 - v2) ^ 2)))
23  }
24
25  # Compute the euclidean distance between two vectors that may contain
        question
26  # mark elements in a linear way.
27  distance_lin <- function(v1, v2){
28    s <- 0
29    for(i in 1:length(v1)){
30      if(v1[i] != '?' & v2[i] != '?'){
31        # Only elements that are not question marks contribute to the
              distance.
32        s <- s + ((as.numeric(v1[i]) - as.numeric(v2[i])) ^ 2)
33      }
34    }
35    return(sqrt(s))
36  }
37
38
39  # Compute the modified nearest neighbour of every element in the
        dataset.
40  nn_data <- clean_data
41  for(i in 1:nrow(clean_data)){
42    min_distance <- -1
43    min_index <- -1
44
45    for(j in 1:nrow(clean_data)){
46      if(i != j){
47        d <- distance(clean_data[i,], clean_data[j,])
48        if(min_distance == -1 | d < min_distance){
49          min_distance <- d
50          min_index <- j
51        }
52      }
53    }
54
55    nn_data[i,] <- clean_data[min_index,]
56  }
```

5.2 Conclusions

Two different implementations have been tried for this exercise, one that computes the distance between two vectors with some question mark elements in a linear way (`distance_lin`) and other that does the same but with vectorial operations (`distance_vec`).

Unfortunately, the linear code that uses this distance functions and other `R` specific instructions to apply the function to matrix columns, like `outer`, were so slow during the execution that it was impossible to obtain results.