CS 199

HW3 - Predicting Crime Rates

April 8, 2014

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

Contents

1 Introduction

This assignment consisted on the implementation of several *regression* methods to try to predict the rate of violent crimes per population of several cities of the United States.

For that purpose, a dataset from the *UC Irvine* machine learning repository was provided. The programming language used have been R, and this report have been typeset using LaTeX.

2 Removing variables with missing values

2.1 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]
```

3 Basics - Linear regression

3.1 Implementation

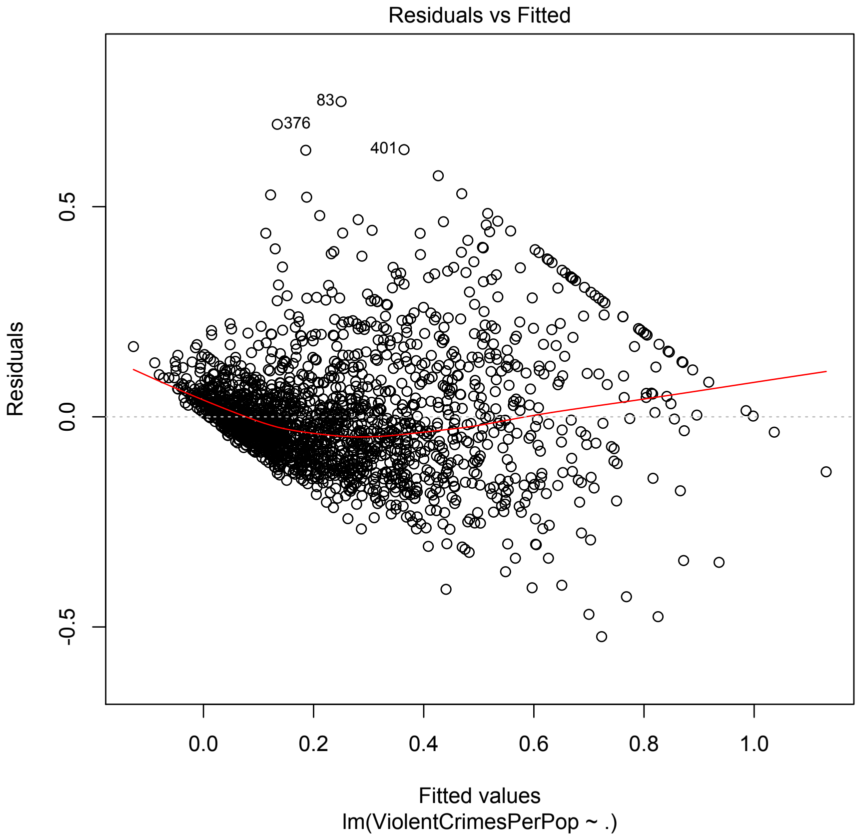
```
# Compute the regression with the data free of unknowns.
linear_regr <- lm(ViolentCrimesPerPop ~ ., data=no_unkw_data)

# 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.
```

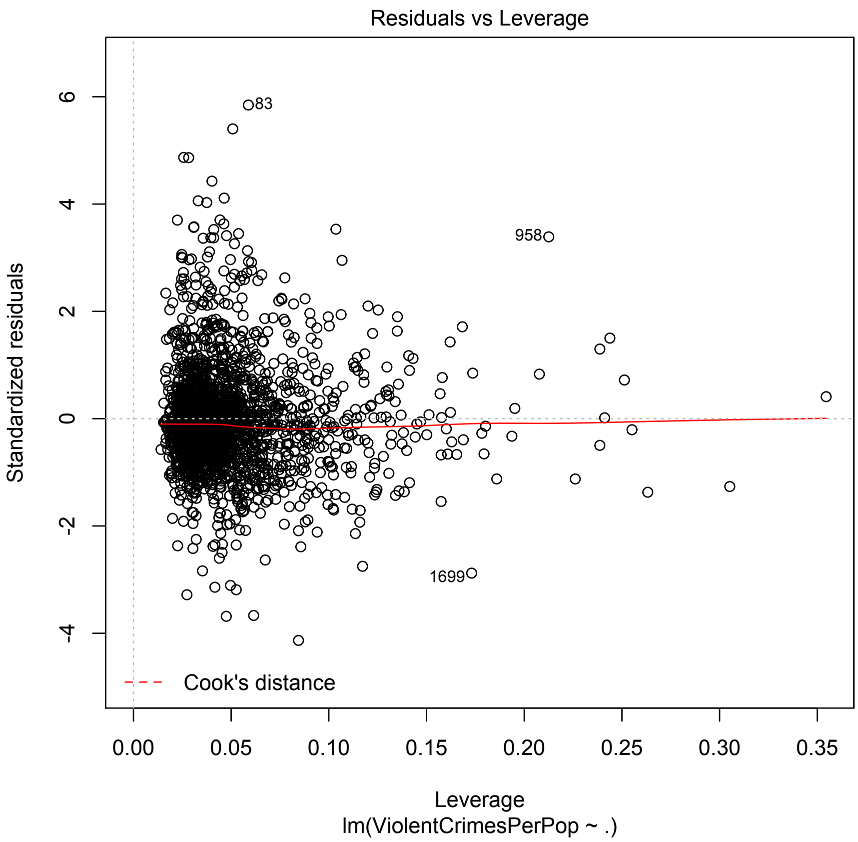
```
residuals_test <- test_data$ViolentCrimesPerPop - predict(linear_regr_test, test_data)
plot(test_data$ViolentCrimesPerPop, residuals_test)

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
# a very small number.
boxcox_data <- no_unkw_data
boxcox_data$ViolentCrimesPerPop[ which(boxcox_data$ViolentCrimesPerPop == 0) ] <- 1e-100

# Apply Box-Cox and get a list of the lamdbas and their log likelihood and obtain
# the value with the highest likelihood.
linear_regr_mod <- lm(ViolentCrimesPerPop ~ ., data=boxcox_data)
lambdas <- boxcox(linear_regr_mod, plotit=F)
max_lambda <- lambdas$x[ which(lambdas$y == max(lambdas$y)) ]

# Change the very low values to zero again.
boxcox_data$ViolentCrimesPerPop[which(boxcox_data$ViolentCrimesPerPop == 1e-100)] <- 0

# Transform the data with the given value of lambda.
if (max_lambda == 0) {
  boxcox_data$ViolentCrimesPerPop <- log(boxcox_data$ViolentCrimesPerPop)
} else {
  boxcox_data$ViolentCrimesPerPop <- (boxcox_data$ViolentCrimesPerPop^max_lambda - 1)/max_lambda
}

# Compute the linear regression again and plot it.
boxcox_regr <- lm(ViolentCrimesPerPop ~ ., data=boxcox_data)
box_residuals_test <- sum((boxcox_regr$residuals - mean(boxcox_regr$residuals)) ^ 2) / length(boxcox_regr$residuals)
printf(" - Mean-squared error on the Boxcox all data: %.2e", box_residuals_test)
plot(boxcox_regr)
```

3.2 Results

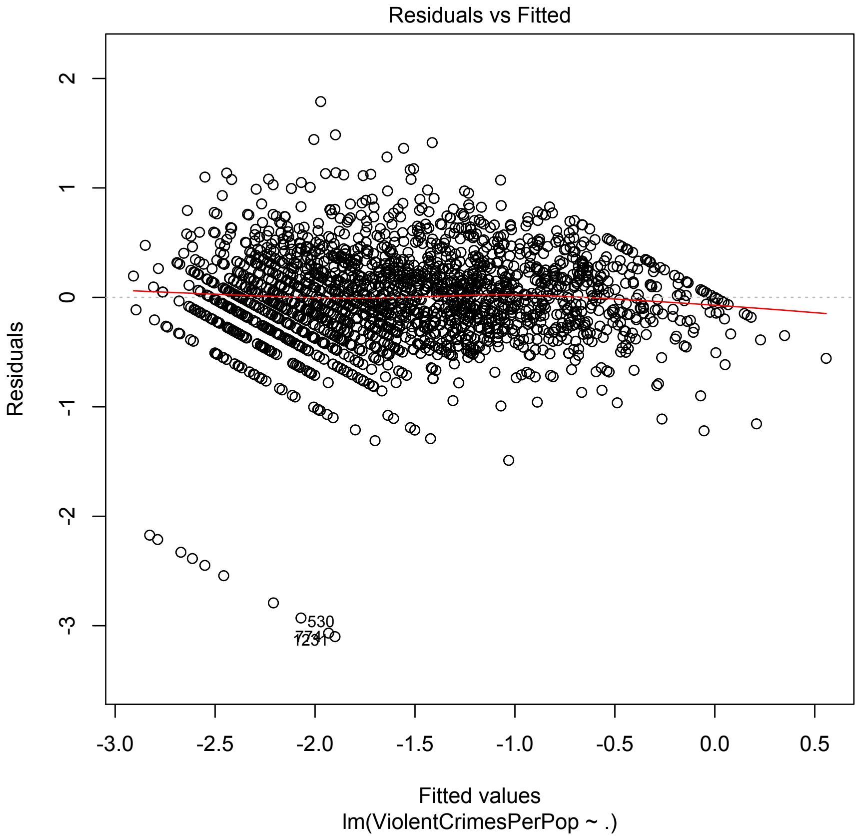
- Mean-squared error on the *whole data*: **1.66e-02**
- Mean-squared error on the *test data* (20%): **1.88e-02**
- Mean-squared error on the *Box-Cox transformed data*: **1.85e-01**

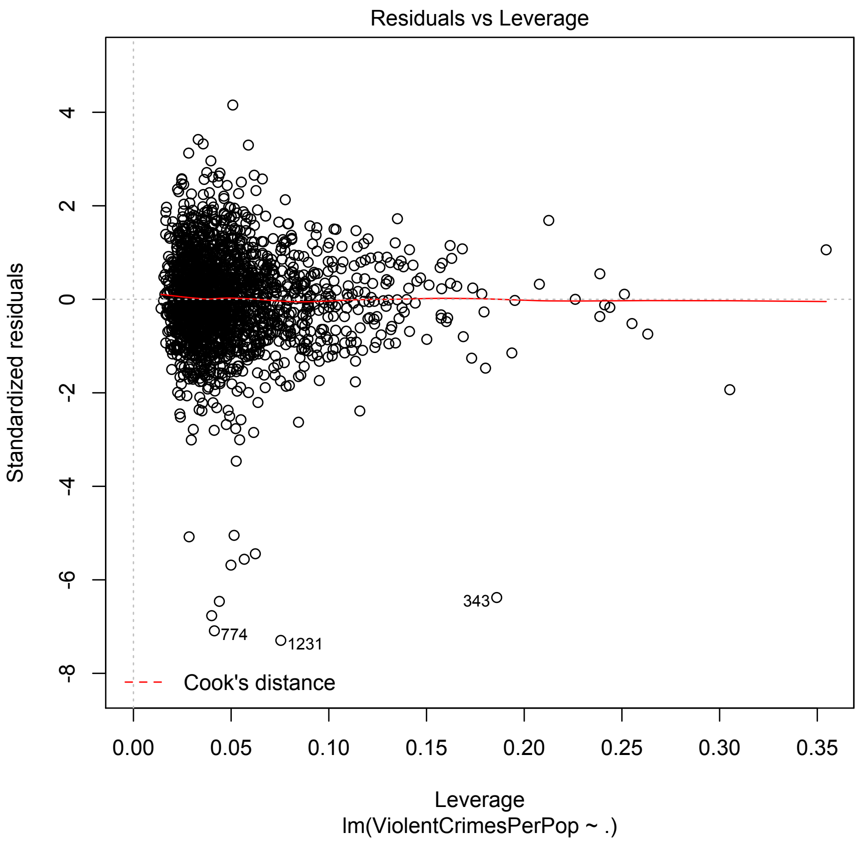
3.2.1 Standard data





3.2.2 Box-Cox transformed data





3.3 Conclusions

3.3.1 Standard data

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.

3.3.2 Box-Cox transformed data

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.

4 Basics - Nearest Neighbor regression

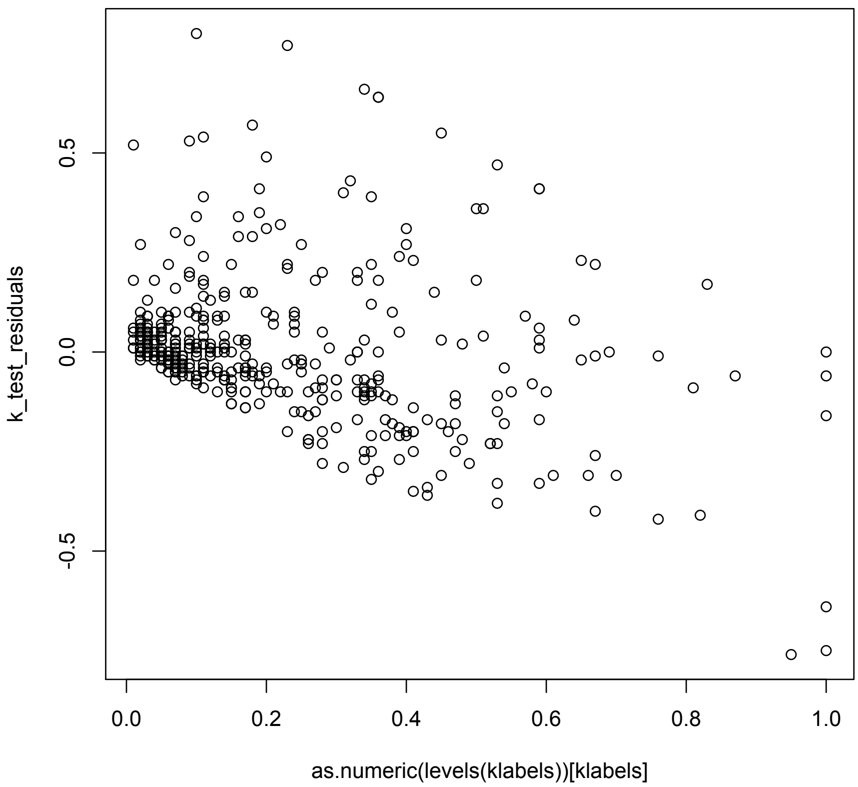
4.1 Implementation

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

4.2 Results

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



4.3 Conclusions

The Mean-squared error on k-nearest neighbor regression is three times as large as the linear regression. The residuals graph looked as though the regression was far worse.

This data seams to be relatively spastic. Thus it dose not very well with nearest neighbor regressions.

Scaling the data changed nothing. I am guessing that the data set was prepossessed.

5 Dealing with missing values

5.1 Implementation

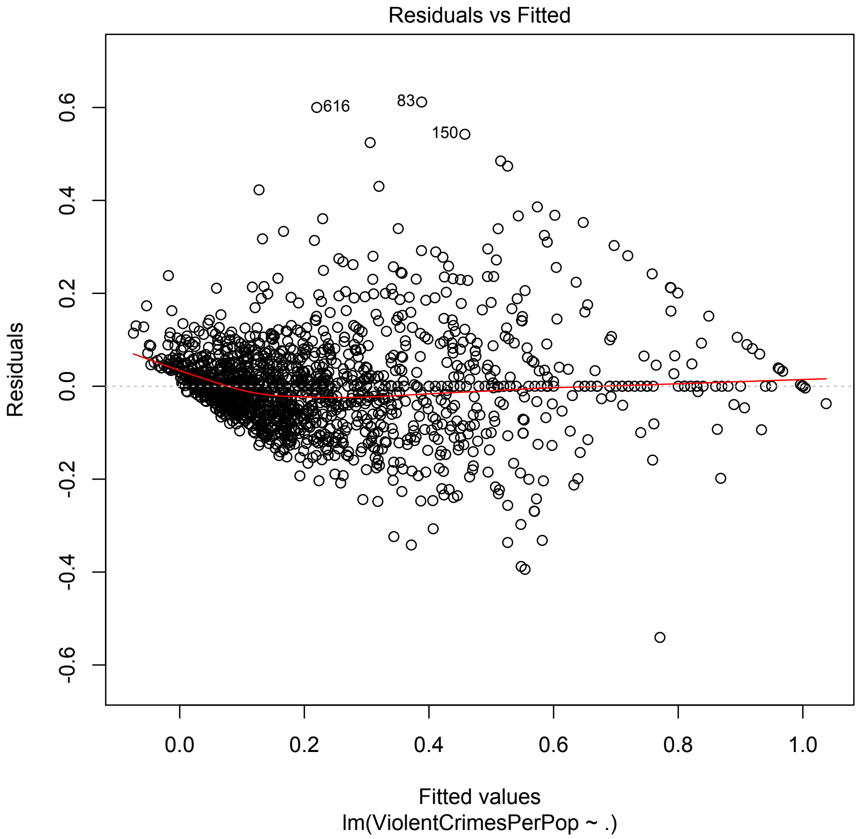
```
# Impute unknown values by covering all the columns with unknowns.
interpolated_data <- clean_data
for(i in attr_with_unkw) {
    # Divide data by rows in two sets: one with the samples that
        contain a question
    # mark in that column and one with the rest.
    unk_indices <- which(clean_data[,i] == '?')
    i_unk_data <- no_unkw_data[unk_indices, ]
    i_no_unk_data <- no_unkw_data[-unk_indices, ]

    # Compute the nearest neighbours in the set of elements with
        question mark of
    # the elements with a question mark in that column.
    i_klabels <- knn(i_no_unk_data,
                     i_unk_data,
                     i_no_unk_data$ViolentCrimesPerPop, k = 1,
                     prob = FALSE, algorithm=c("kd_tree"))
    indices <- attr(i_klabels, "nn.index")
    nn_subs <- clean_data[-unk_indices, ][indices,]

    # Substitute the question mark values by the value of the nearest
        neighbour.
    interpolated_data[unk_indices, i] <- nn_subs[,i]
}

# Split off new test data.
folds <- cvFolds(nrow(interpolated_data), K=5)
train_data <- interpolated_data[folds$subsets[folds$which != 1], ]
test_data  <- interpolated_data[folds$subsets[folds$which == 1], ]

# Compute a linear regression again and evaluate it with mean-squares.
printf("Linear Regression (imputed missing values):")
linear_regr_test <- lm(ViolentCrimesPerPop ~ ., data=train_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)

# Also, compute a nearest neighbour regression and evaluate it.
printf("Nearest Neighbours (imputed missing values):")
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"))
```

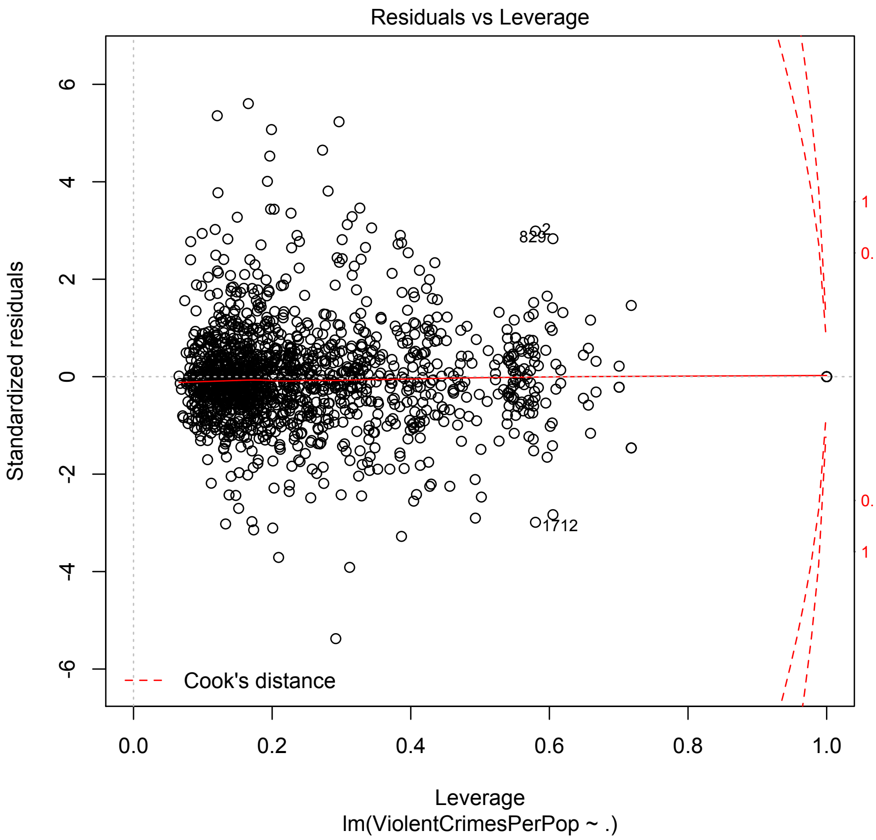
```
k_test_residuals <- test_data$ViolentCrimesPerPop - as.numeric(levels(
    klabels))[klabels]
plot(as.numeric(levels(klabels))[klabels], k_test_residuals)

# Compute the mean-squared error and print the result.
mse_residuals_knn_test <- sum((k_test_residuals - mean(k_test_residuals
    )) ^ 2) /
     length(k_test_residuals)
printf(" - Mean-squared error on the test data (20%%): %.2e", mse_
    residuals_knn_test)
```

5.2 Results

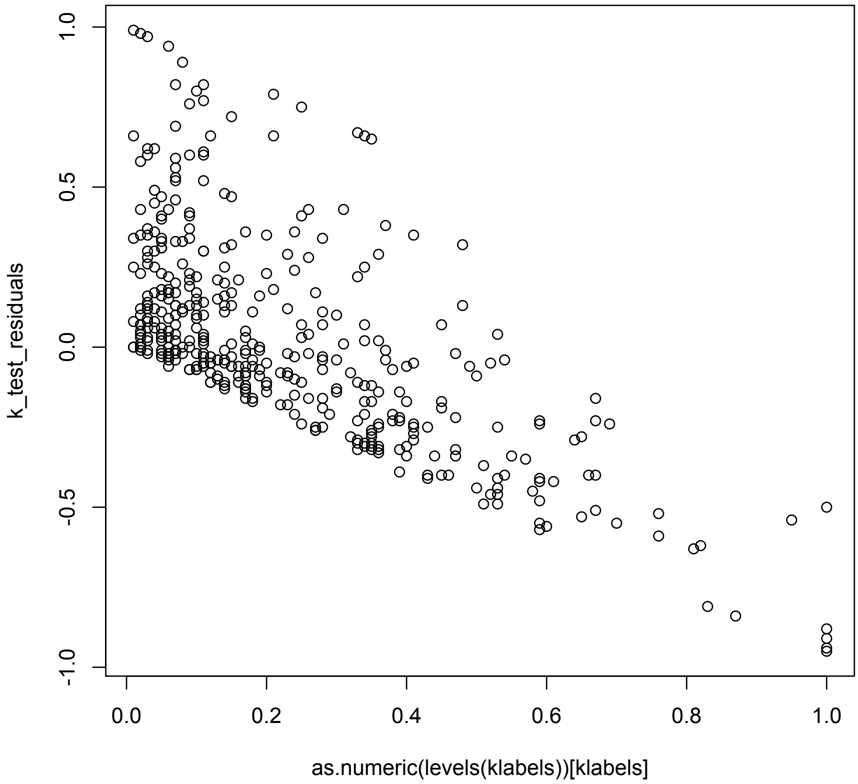
Linear Regression (imputed missing values): - Mean-squared error on the test data (20%): 1.88e-02 Nearest Neighbours (imputed missing values): - Mean-squared error on the test data (20%): 1.00e-01

5.2.1 Linear regression





5.2.2 Nearest Neighbours



5.3 Conclusions

The Mean-squared error of the test sets did not change much for liner regression on this new data. The Mean-square error was far worse for the nearest neighbors.

If Residuals graph looks like it has a slightly better shape but larger residuals. if you look at the leverage graph it is clear that a few points have higher leverage and one point even has a cooks distance of one.

It seams that implementation of missing values lets us have more explanatory variables but add more noise. For linear regression it's a wash but for nearest neighbors the noise is too much.

The additional explanatory variables is not enough to fix our regression and are likely not woth the truble.

6 Modified Nearest Neighbours

6.1 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
  v2[unk_indices] <- 0

  # Ensure vectors type is numeric.
  v1 <- as.numeric(v1)
  v2 <- as.numeric(v2)

  # Return the euclidian distance.
  return(sqrt(sum((v1 - v2) ^ 2)))
}

# Compute the euclidean distance between two vectors that may contain question
# mark elements in a linear way.
distance_lin <- function(v1, v2){
  s <- 0
  for(i in 1:length(v1)){
    if(v1[i] != '?' & v2[i] != '?'){
      # Only elements that are not question marks contribute to the distance.
      s <- s + ((as.numeric(v1[i]) - as.numeric(v2[i])) ^ 2)
    }
  }
  return(sqrt(s))
}


# Compute the modified nearest neighbour of every element in the dataset.
nn_data <- clean_data
for(i in 1:nrow(clean_data)){
  min_distance <- -1
  min_index <- -1

  for(j in 1:nrow(clean_data)){
    if(i != j){
      d <- distance(clean_data[i,], clean_data[j,])
      if(min_distance == -1 | d < min_distance){
        min_distance <- d
        min_index <- j
      }
    }
  }

  nn_data[i,] <- clean_data[min_index,]
}
```

6.2 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.