K-nn

Rodrigo Falcão

2/25/2021

O conjunto de treino e teste foi dividido seguindo o 5-fold cross-validation e o tempo de execução foi medido em segundos

Primeira Base de dados (mostrando o "head" da base):

loc	v(g)	ev(g)	iv(G)	N	V	L	D	I	E	B
1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	1.30
1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	1.00
91.0	9.0	3.0	2.0	318.0	2089.21	0.04	27.68	75.47	57833.24	0.70
109.0	21.0	5.0	18.0	381.0	2547.56	0.04	28.37	89.79	72282.68	0.85
505.0	106.0	41.0	82.0	2339.0	20696.93	0.01	75.93	272.58	1571506.88	6.90
107.0	25.0	7.0	14.0	619.0	4282.78	0.02	52.91	80.95	226588.75	1.43

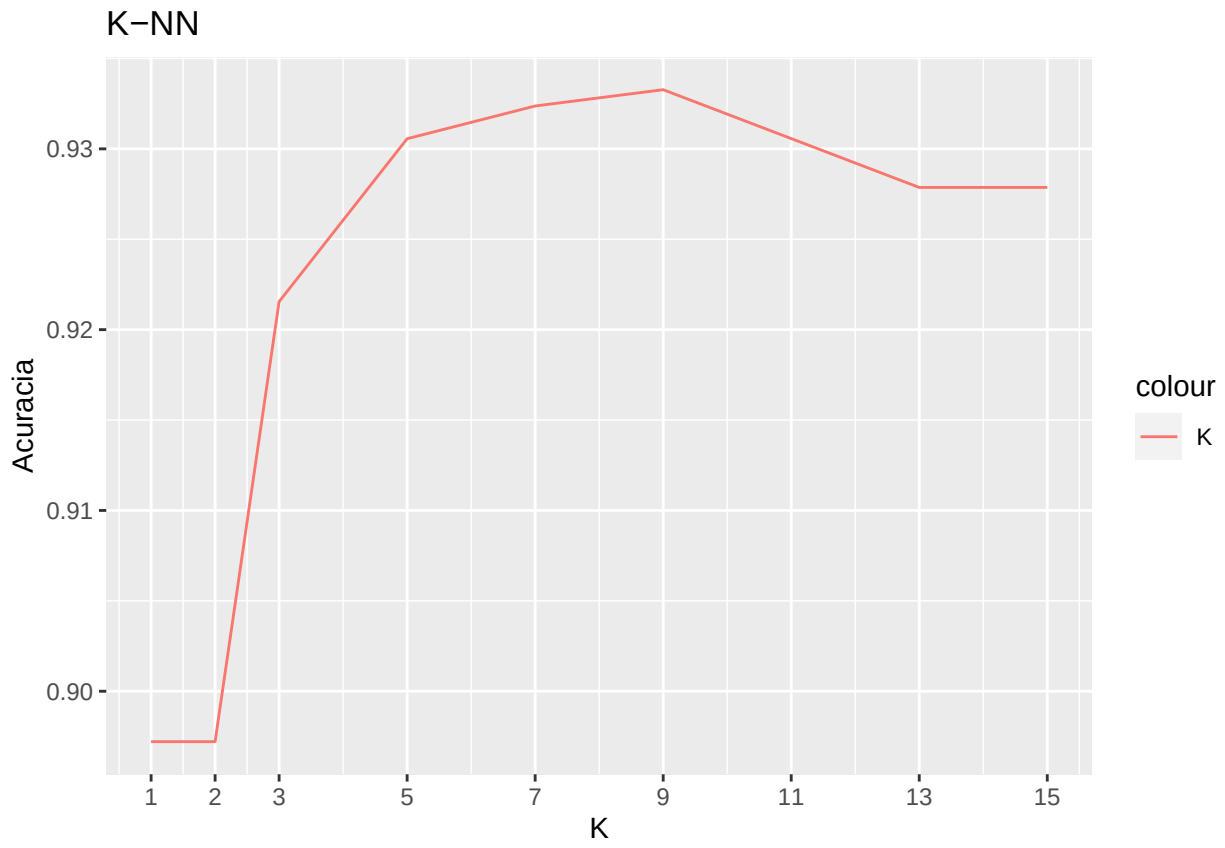
B	T	lOCode	lOComment	locCodeAndComment
1.30	1.30	2	2	2
1.00	1.00	1	1	1
0.70	3212.96	80	44	11
0.85	4015.70	97	41	12
6.90	87305.94	457	71	48
1.43	12588.26	103	32	4

lOBlank	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount	defects
2	1.2	1.2	1.2	1.2	1.4	FALSE
1	1.0	1.0	1.0	1.0	1.0	TRUE
31	29.0	66.0	192.0	126.0	17.0	TRUE
24	28.0	75.0	229.0	152.0	38.0	TRUE
49	64.0	397.0	1397.0	942.0	178.0	TRUE
39	35.0	86.0	359.0	260.0	40.0	TRUE

K-NN:

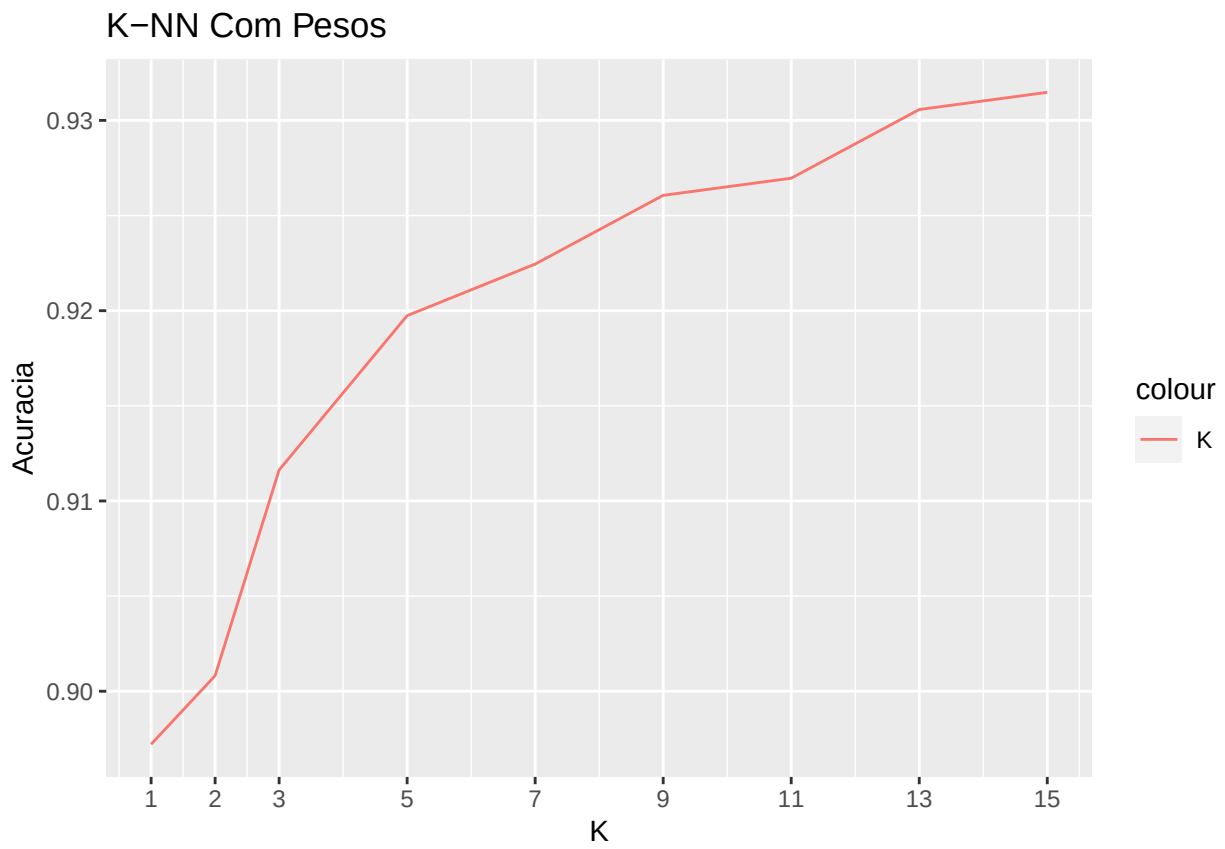
K	Acuracia	Tempo	Razao
1	0.8972117	575.00	0.0015604
2	0.8972117	573.58	0.0015642
3	0.9215482	573.01	0.0016083
5	0.9305613	571.03	0.0016296

K	Acuracia	Tempo	Razao
7	0.9323713	571.53	0.0016314
9	0.9332722	572.66	0.0016297
11	0.9305695	571.30	0.0016289
13	0.9278668	571.09	0.0016247
15	0.9278668	572.58	0.0016205



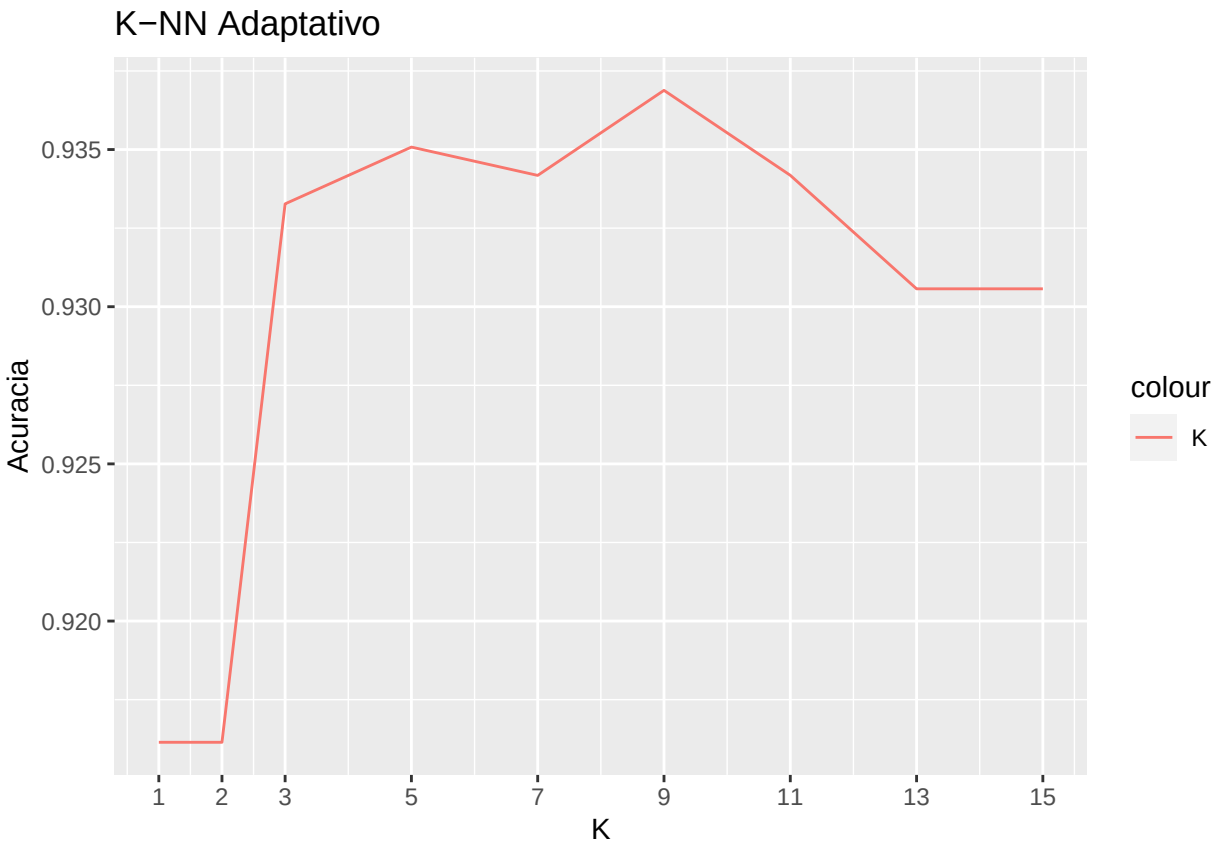
K-NN com pesos:

K	Acuracia	Tempo	Razao
1	0.8972117	575.40	0.0015593
2	0.9008153	575.44	0.0015654
3	0.9116261	576.50	0.0015813
5	0.9197383	574.20	0.0016018
7	0.9224491	575.02	0.0016042
9	0.9260609	575.63	0.0016088
11	0.9269618	574.78	0.0016127
13	0.9305695	574.42	0.0016200
15	0.9314745	574.61	0.0016211



K-NN Adaptativo:

K	Acuracia	Tempo	Razao
1	0.9161428	1127.62	0.0008125
2	0.9161428	1121.28	0.0008171
3	0.9332722	1118.44	0.0008344
5	0.9350781	1119.31	0.0008354
7	0.9341772	1119.69	0.0008343
9	0.9368839	1119.31	0.0008370
11	0.9341812	1120.22	0.0008339
13	0.9305695	1119.66	0.0008311
15	0.9305695	1120.08	0.0008308

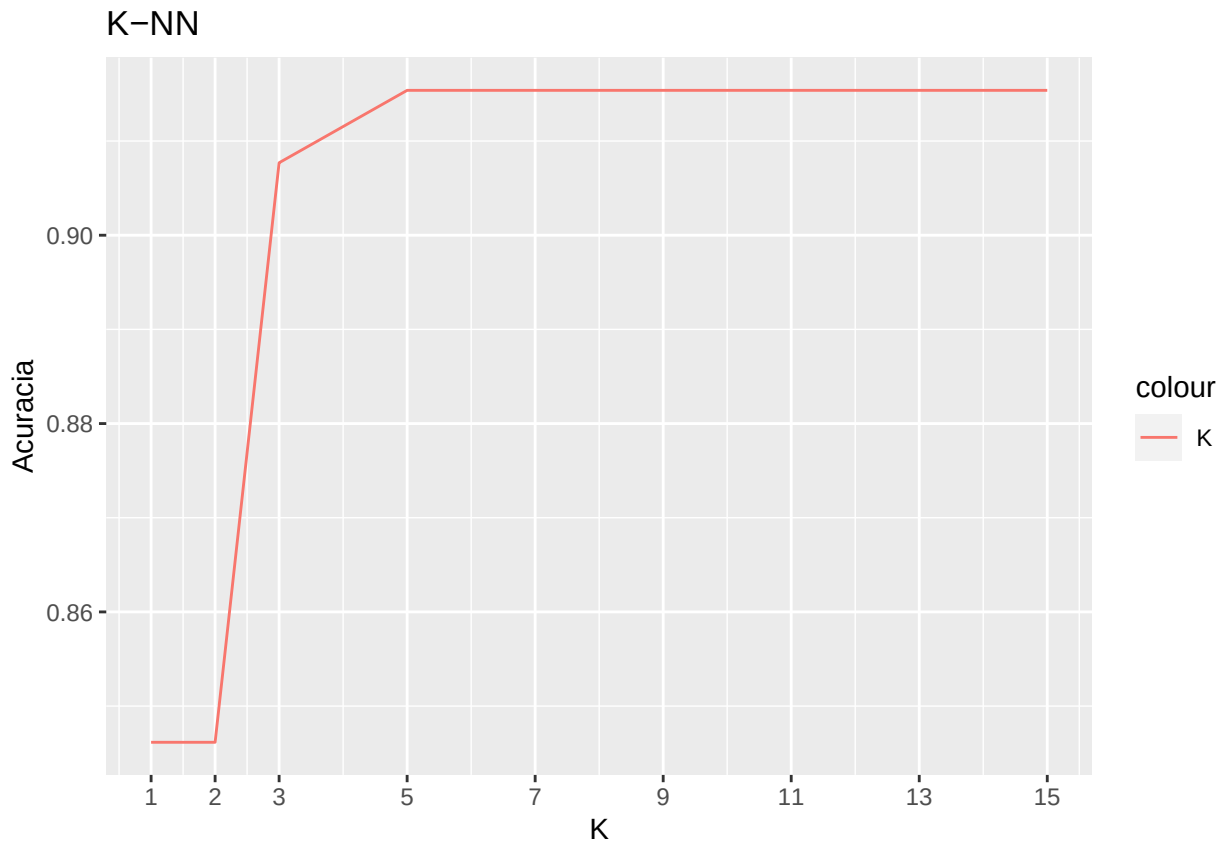


Segunda Base de dados (mostrando o "head" da base):

LOC6_0	LOC6_1	Added_LoC	Del_LoC	Diff_Block	Mod_Rate	Mod_Know	ReusedLoC	Faulty6_1
127	126	6	7	2	10	1	120	0
458	441	50	67	17	23	1	391	0
182	178	10	14	3	13	1	168	0
270	270	14	14	3	10	1	256	0
107	104	11	14	4	21	1	93	0
892	869	40	63	16	11	1	829	0

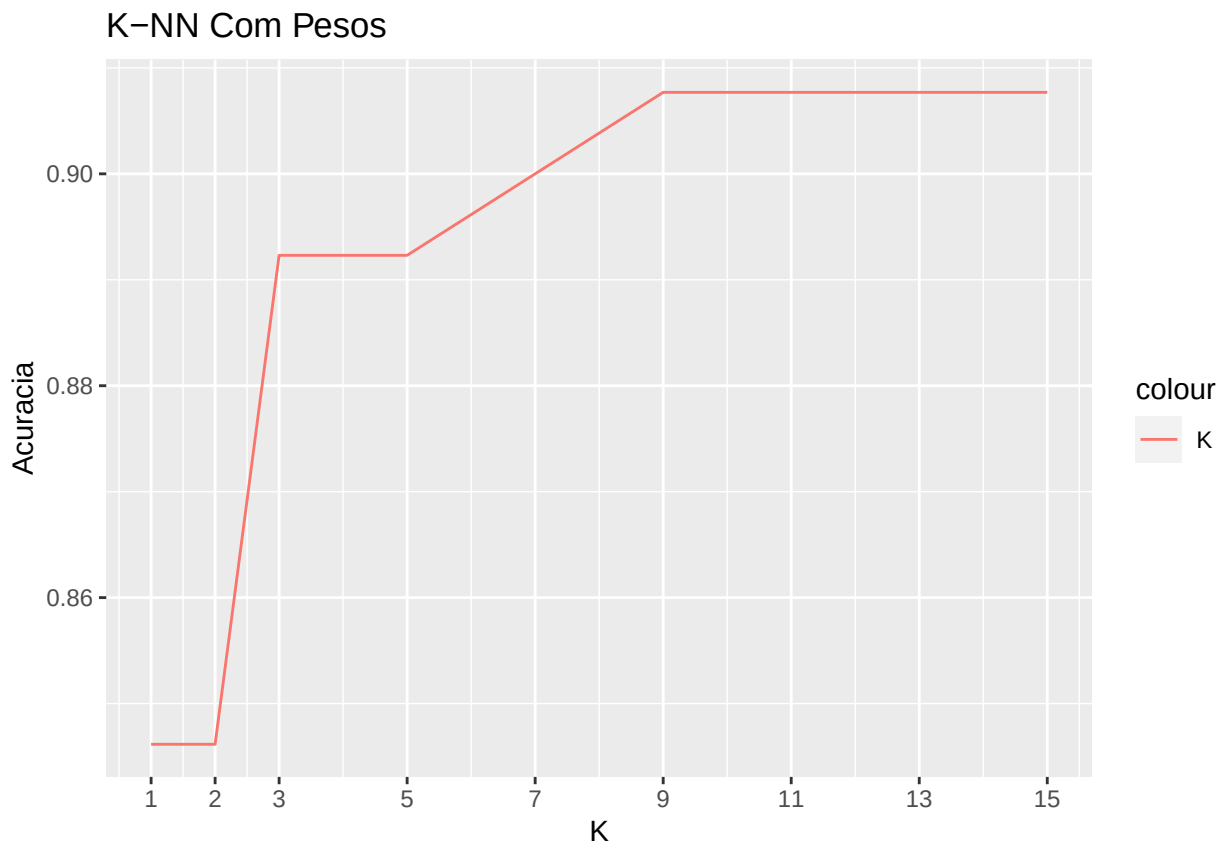
K-NN:

K	Acuracia	Tempo	Razao
1	0.8461538	4.65	0.1819686
2	0.8461538	4.53	0.1867889
3	0.9076923	4.60	0.1973244
5	0.9153846	4.48	0.2043269
7	0.9153846	4.56	0.2007422
9	0.9153846	4.58	0.1998656
11	0.9153846	4.60	0.1989967
13	0.9153846	4.54	0.2016266
15	0.9153846	4.61	0.1985650



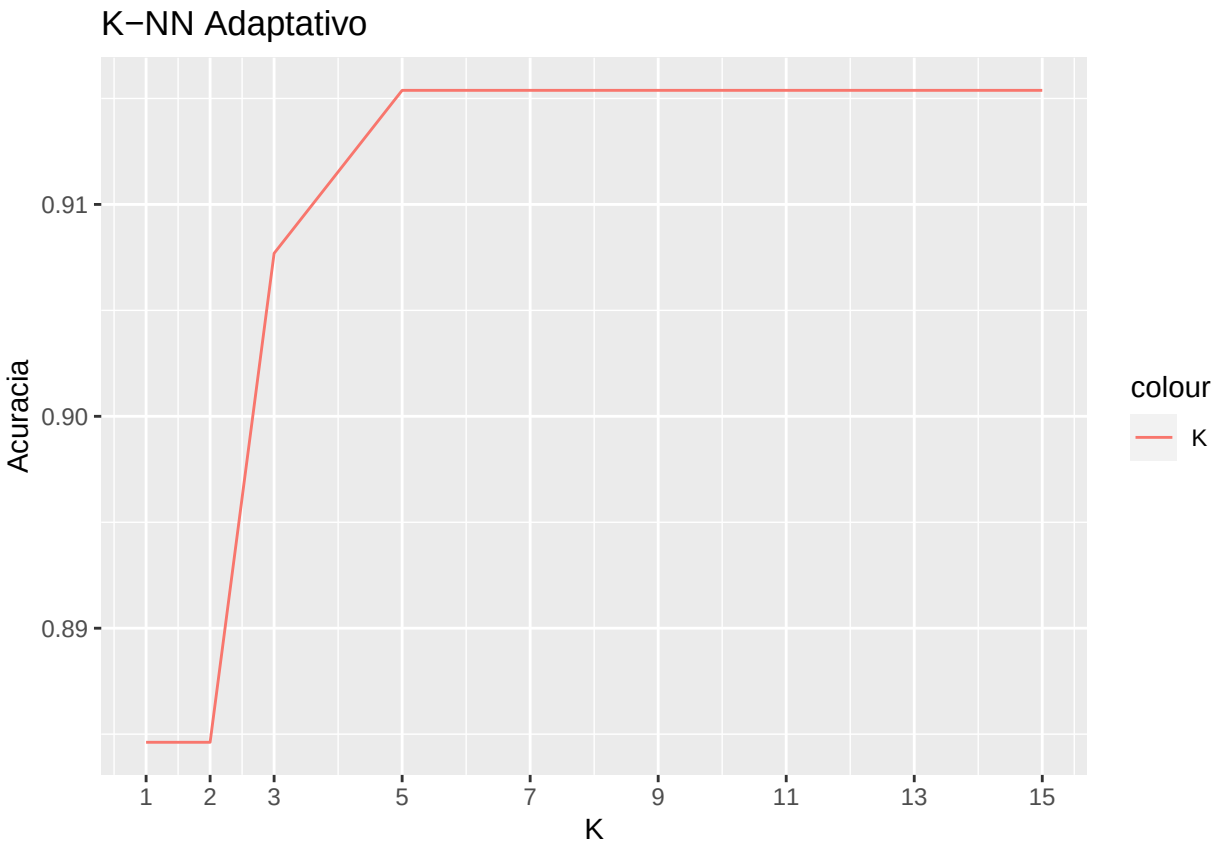
K-NN com pesos:

K	Acuracia	Tempo	Razao
1	0.8461538	4.71	0.1796505
2	0.8461538	4.72	0.1792699
3	0.8923077	4.72	0.1890482
5	0.8923077	4.69	0.1902575
7	0.9000000	4.78	0.1882845
9	0.9076923	4.67	0.1943667
11	0.9076923	4.71	0.1927160
13	0.9076923	4.70	0.1931260
15	0.9076923	4.80	0.1891026



K-NN Adaptativo:

K	Acuracia	Tempo	Razao
1	0.8846154	12.18	0.0726285
2	0.8846154	12.28	0.0720371
3	0.9076923	12.11	0.0749539
5	0.9153846	12.14	0.0754024
7	0.9153846	12.01	0.0762185
9	0.9153846	12.07	0.0758397
11	0.9153846	12.07	0.0758397
13	0.9153846	12.08	0.0757769
15	0.9153846	12.16	0.0752783



Análise de resultados

A pontuação para o melhor valor de K é baseada na razão "Acurácia/Tempo", na qual a acurácia corresponde a porcentagem de acerto para cada valor de K e o tempo corresponde à soma do tempo de teste e treinamento para cada valor de K. Tal pontuação corresponde ao melhor valor de K pois a razão garante que: quanto maior a acurácia maior a razão e quanto menor o tempo maior a razão.

Dessa forma, o melhor valor de K para cada algoritmo na primeira base de dados é:

- K-NN: 7
- K-NN com pesos: 15
- K-NN adaptativo: 9

Da mesma forma, o melhor valor de K para cada algoritmo na segunda base de dados é:

- K-NN: 5
- K-NN com pesos: 9
- K-NN adaptativo: 7

K-NN implementado (em R)

```
library(foreign)
library(caTools)

moda <- function(v) {
  uniqv <- unique(v)
  return(uniqv[which.max(tabulate(match(v, uniqv)))])
}


divideInstances <- function(df, fraction){
  smp_size <- floor(fraction * nrow(df))

  set.seed(123)
  train_ind <- sample(seq_len(nrow(df)), size = smp_size)

  train <- df[train_ind, ]
  test <- df[-train_ind, ]
  return(list(train, test))
}

k_proximos <- function(instance, examples, k, classCol){
  x <- examples[,-classCol]
  y <- examples[,classCol]
  dists <- c()
  classe <- c()
  for (i in 1:nrow(x)) {
    dists <- append(dists, dist(rbind(x[i,], instance)))
    classe <- append(classe, y[i])
  }
  df <- data.frame(classe = classe, dists = dists)
  df <- df[order(df$dists),]
  proximos <- as.vector(as.numeric(rownames(df[1:k,])))
  candidates <- y[as.numeric(proximos)]
  result <- moda(candidates)
  return(result)
}


calculateKnnAccuracy <- function(k, test, train, classCol){
  predictions <- apply(test[,-classCol], 1, function(x) k_proximos(x, train, k, classCol))
  testy <- test[,classCol]
  dif <- predictions == testy
  return(sum(dif)/length(predictions))
}
```

K-NN com pesos implementado (em R)

```
library(foreign)
library(caTools)

moda <- function(v) {
  uniqv <- unique(v)
  return(uniqv[which.max(tabulate(match(v, uniqv)))])
}

get_weighted <- function(labels,weights) {
  dset = data.frame(labels=factor(labels),
                    weights=weights)
  scores = aggregate(. ~ labels, dset, sum)
  result = as.character(scores$labels[which.max(scores$weights)])
  return(result)
}

divideInstances <- function(df, fraction){
  smp_size <- floor(fraction * nrow(df))

  set.seed(123)
  train_ind <- sample(seq_len(nrow(df)), size = smp_size)

  train <- df[train_ind, ]
  test <- df[-train_ind, ]
  return(list(train, test))
}

pesosk_proximos <- function(instance, examples, k, classCol){
  x <- examples[,-classCol]
  y <- examples[,classCol]
  dists <- c()
  classe <- c()
  for (i in 1:nrow(x)) {
    dists <- append(dists, dist(rbind(x[i,], instance)))
    classe <- append(classe, y[i])
  }
  if (any(dists==0)) {
    ind <- which(dists==0)
    res <- moda(y[ind])
    return(res)
  }else {
    proximidade <- 1/dists
    df <- data.frame(classe = classe, dists = dists)
    df <- df[order(df$dists),]
    proximos <- as.vector(as.numeric(rownames(df[1:k,])))
    candidates <- y[as.numeric(proximos)]
    pesos <- proximidade[as.numeric(proximos)]
    result <- get_weighted(candidates,pesos)
    return(result)
  }
}


calculatePesoKnnAccuracy <- function(k, test, train, classCol){
  predictions <- apply(test[,-classCol], 1, function(x) pesosk_proximos(x, train, k, classCol))
  testy <- test[,classCol]
  dif <- predictions == testy
  return(sum(dif)/length(predictions))
}
```

K-NN adaptativo implementado (em R)

```
library(foreign)
library(caTools)

moda <- function(v) {
  uniqv <- unique(v)
  return(uniqv[which.max(tabulate(match(v, uniqv)))])
}

divideInstances <- function(df, fraction){
  smp_size <- floor(fraction * nrow(df))

  set.seed(123)
  train_ind <- sample(seq_len(nrow(df)), size = smp_size)

  train <- df[train_ind, ]
  test <- df[-train_ind, ]
  return(list(train, test))
}

adaptivek_proximos <- function(instance, train, k, classCol, raios){
  x <- train[,-classCol]
  y <- train[,classCol]
  dists <- c()
  classe <- c()
  for (i in 1:nrow(x)) {
    dists <- append(dists, dist(rbind(x[i,], instance))/raios[i])
    classe <- append(classe, y[i])
  }
  df <- data.frame(classe = classe, dists = dists)
  df <- df[order(df$dists),]
  proximos <- as.vector(as.numeric(rownames(df[1:k,])))
  candidates <- y[as.numeric(proximos)]
  result <- moda(candidates)
  return(result)
}

adaptiveRule <- function(train, colClass){
  radius <- c()
  for (i in 1:nrow(train)) {
    aux <- c()
    for (j in 1:nrow(train)) {
      if(train[i, colClass] != train[j, colClass]){
        distancia <- dist(rbind(train[i, -colClass], train[j, -colClass])) - 0.00000000001
        if(distancia >= 0){
          aux <- append(aux, distancia)
        }
      }
    }
    radius <- append(radius, min(aux))
  }
  return(radius)
}

adaptiveKnnAccuracy <- function(k, test, train, classCol){
  raios <- adaptiveRule(train, classCol)
  predictions <- apply(test[,-classCol], 1, function(x) adaptivek_proximos(x, train, k, classCol, raios))
  testy <- test[,classCol]
  dif <- predictions == testy
  return(sum(dif)/length(predictions))
}
```

K-fold implementado em R (para o a base de dados "df", correspondente à primeira base de dados)

```
folds <- rep_len(1:nrFolds, nrow(df))
accuracy <- c()
tempo <-c()
kValue <- c(1,2,3,5,7,9,11,13,15)
for (i in kValue) {

  start <- proc.time()
  auxAccu <- c()
  for(k in 1:nrFolds) {
    fold <- which(folds == k)
    train <- df[-fold,]
    test <- df[fold,]

    result <- calculateKnnAccuracy(i, test, train, 22)


    auxAccu <- append(auxAccu, result)

  }
  end <- proc.time()
  tempo <- append(tempo, (end - start)[[3]])
  accuracy <- append(accuracy, mean(auxAccu))
}
```