Lab 02 - Aprendizagem de Maquina

Helena Regina Salomé DEspindula

2024-08-31

```
library(plotly)
```

```
## Carregando pacotes exigidos: ggplot2
```

```
##
## Anexando pacote: 'plotly'
```

```
## O seguinte objeto é mascarado por 'package:ggplot2':
##
##     last_plot
```

```
## O seguinte objeto é mascarado por 'package:stats':
##
##     filter
```

```
## O seguinte objeto é mascarado por 'package:graphics':
##
##     layout
```

```
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks plotly::filter(), stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Classificadores

Codigo base, com leves modificações da aluna pode ser encontrado ao final desse relatorio.

Regressão Logistica (LR)

Utilizando a linha *clf = linear_model.LogisticRegression()*

```
[...]

Training size...  58000
CLF score =  0.9474
--- 7.33170080 seconds ---
Scores
[0.7393, 0.8624, 0.8943, 0.9000, 0.9129, 0.9197, 0.9242, 0.9283, 0.9280, 0.9314, 0.9335, 0.9362, 0.9352
Tempo
[0.158, 0.2674, 0.5048, 0.397, 0.3819, 0.4371, 0.5025, 0.7137, 0.6554, 1.2889, 1.8421, 1.9517, 0.9907,
```

```
Loading data...
Training size...  58000
CLF score =  0.9474
[[1950   14    8    0   20    4   18    0   20    4]
 [   0 1954   30    6    6   14    0   20    8    0]
 [   0   30 1902   12    4    6    2   12   26    0]
 [   2   46   18 1850    0   24    8   36   16    4]
 [   0    8    6    0 1924    4    8    6   16    4]
 [   0   44    0   12    2 1864    6    6   12    0]
 [   2   28   14    0   10    2 1922    0    4    0]
 [   0   14    4    0    6    0    0 1936    4   76]
 [  54   46    6    4   58   28    8    6 1760   10]
 [   2   18    2    4   28   16    0   44    2 1886]]
```

K-nearest-neighbor (KNN)

Utilizando a linha *clf = KNeighborsClassifier(n_neighbors=3, metric=euclidean)*

```
[...]

Training size...  58000
CLF score =  0.9557
--- 52.10682297 seconds ---
Scores
[0.7532, 0.8995, 0.9194, 0.9252, 0.9354, 0.9333, 0.9363, 0.9369, 0.9367, 0.9368, 0.9394, 0.9400, 0.9403
Tempo
[4.2682, 4.5013, 6.571, 5.8672, 7.9289, 7.6253, 9.5135, 9.1288, 10.3448, 11.729, 12.4124, 13.1689, 13.8
```

```
Loading data...
Training size...  58000
CLF score =  0.9474
[[1982    6    2    2    8    6    8    0   14   10]
 [   4 1970   30    4    0   10    2   10    8    0]
 [   0   28 1908   12    2    8    2   14   14    6]
 [   4   32   14 1888    0   14    4   24   14   10]
 [   4   10    4    2 1932    6    0    4    4   10]
 [   0   38    2   22    0 1860   10    0   12    2]
```

```
 [   6   12   10    4    8   10 1928    0    2    2]
 [   0   22    8    6   12    0    0 1896    0   96]
 [  20   34    4   12   16   20    6    2 1848   18]
 [   2   16    4   12   18   14    0   20   14 1902]]
```

Gaussiana NB (GNB)

Utilizando a linha *clf = GaussianNB()*

```
[...]

Training size...  58000
CLF score =  0.9121
--- 0.67942882 seconds ---
Scores
[0.6558, 0.8556, 0.8691, 0.8796, 0.8971, 0.8941, 0.8958, 0.8948, 0.8958, 0.9001, 0.8988, 0.9001, 0.8984
Tempo
[0.4754, 0.4584, 0.4665, 0.4664, 0.4654, 0.479, 0.464, 0.4882, 0.4865, 0.4978, 0.4741, 0.4852, 0.5011, 
```

```
Loading data...
Training size...  58000
CLF score =  0.9121
[[1742    0   16    0    8    4  102    0  158    8]
 [   0 1916   36    0   16   10    0   36   18    6]
 [   2   14 1804   76    6    2    0   20   46   24]
 [   6    8   36 1878    0   14    2   30   18   12]
 [   2    4    2    0 1804   14   30    8   80   32]
 [   6   12    2   60    0 1814   12    2   14   24]
 [   4    4   22    0   10   20 1856    0   52   14]
 [   0   14   20   10    8    6    0 1956   14   12]
 [ 104    6   22    6   30   24   28    2 1720   38]
 [   2    6    4   16   24   10    0   74  114 1752]]
```

Linear discriminant analysis (LDA)

Utilizando a linha *clf = LinearDiscriminantAnalysis()*

```
[...]

Training size...  58000
CLF score =  0.9354
--- 1.65640712 seconds ---
Scores
[0.7908, 0.9055, 0.9169, 0.9182, 0.9275, 0.9295, 0.9305, 0.9325, 0.9319, 0.9316, 0.9330, 0.9338, 0.9336
Tempo
[0.1862, 1.6473, 0.1882, 0.2244, 0.223, 0.2709, 0.2105, 0.2381, 0.2632, 0.2726, 0.2896, 0.3073, 0.3339,
```

```
Loading data...
Training size...  58000
CLF score =  0.9354
```

```
[[1898   26    8    2   10   16   14    4   56    4]
 [   2 1926   24    4   18   24    0   28    8    4]
 [   2   14 1836   12   16   10    6   62   32    4]
 [   0   18   16 1842    4   42    6   44   26    6]
 [   2   12   12    0 1890   10    0    8   12   30]
 [   0   10    4   18   18 1862   10   10   14    0]
 [   0   14   14    4   10   14 1914    4    6    2]
 [   0   14   14    2   14   12    4 1914    0   66]
 [  32   28   14   10   40   48   10    4 1778   16]
 [   2   18   10    6   14   30    2   44   28 1848]]
```

Analise comparativa

Gerando um Data.frame dos resultados

```
blocos <- 1:58

## scores

lr_score <- c(
  0.7393, 0.8624, 0.8943, 0.9000, 0.9129, 0.9197, 0.9242,
  0.9283, 0.9280, 0.9314, 0.9335, 0.9362, 0.9352, 0.9368,
  0.9372, 0.9370, 0.9388, 0.9395, 0.9404, 0.9392, 0.9392,
  0.9395, 0.9415, 0.9415, 0.9418, 0.9428, 0.9428, 0.9432,
  0.9431, 0.9424, 0.9441, 0.9440, 0.9451, 0.9449, 0.9458,
  0.9455, 0.9459, 0.9447, 0.9457, 0.9458, 0.9455, 0.9452,
  0.9460, 0.9461, 0.9459, 0.9464, 0.9462, 0.9466, 0.9466,
  0.9465, 0.9471, 0.9466, 0.9467, 0.9463, 0.9467, 0.9469,
  0.9465, 0.9474
)
knn_score <- c(
  0.7532, 0.8995, 0.9194, 0.9252, 0.9354, 0.9333, 0.9363,
  0.9369, 0.9367, 0.9368, 0.9394, 0.9400, 0.9403, 0.9425,
  0.9431, 0.9445, 0.9450, 0.9459, 0.9458, 0.9450, 0.9451,
  0.9452, 0.9478, 0.9496, 0.9506, 0.9511, 0.9517, 0.9516,
  0.9518, 0.9513, 0.9516, 0.9516, 0.9531, 0.9527, 0.9536,
  0.9533, 0.9531, 0.9532, 0.9533, 0.9541, 0.9544, 0.9538,
  0.9538, 0.9540, 0.9541, 0.9547, 0.9544, 0.9542, 0.9552,
  0.9557, 0.9560, 0.9560, 0.9555, 0.9558, 0.9553, 0.9556,
  0.9554, 0.9557
)
gnb_score <- c(
  0.6558, 0.8556, 0.8691, 0.8796, 0.8971, 0.8941, 0.8958,
  0.8948, 0.8958, 0.9001, 0.8988, 0.9001, 0.8984, 0.8993,
  0.8995, 0.8977, 0.9017, 0.9004, 0.9018, 0.9031, 0.9078,
  0.9074, 0.9097, 0.9088, 0.9097, 0.9109, 0.9109, 0.9112,
  0.9112, 0.9112, 0.9109, 0.9111, 0.9115, 0.9110, 0.9118,
  0.9115, 0.9105, 0.9106, 0.9109, 0.9113, 0.9106, 0.9108,
  0.9109, 0.9117, 0.9119, 0.9122, 0.9125, 0.9111, 0.9115,
  0.9112, 0.9121, 0.9118, 0.9120, 0.9124, 0.9120, 0.9118,
  0.9119, 0.9121
```

```
)
lda_score <- c(
  0.7908, 0.9055, 0.9169, 0.9182, 0.9275, 0.9295, 0.9305,
  0.9325, 0.9319, 0.9316, 0.9330, 0.9338, 0.9336, 0.9343,
  0.9334, 0.9335, 0.9330, 0.9339, 0.9343, 0.9341, 0.9338,
  0.9334, 0.9334, 0.9336, 0.9344, 0.9349, 0.9355, 0.9349,
  0.9346, 0.9345, 0.9348, 0.9350, 0.9351, 0.9352, 0.9349,
  0.9355, 0.9356, 0.9358, 0.9360, 0.9357, 0.9357, 0.9358,
  0.9351, 0.9353, 0.9349, 0.9350, 0.9348, 0.9352, 0.9353,
  0.9358, 0.9358, 0.9361, 0.9354, 0.9359, 0.9355, 0.9359,
  0.9357, 0.9354
)

## tempo

lr_tempo <- c(
  0.1580, 0.2674, 0.5048, 0.397, 0.3819, 0.4371,
  0.5025, 0.7137, 0.6554, 1.2889, 1.8421, 1.9517,
  0.9907, 1.0192, 1.0998, 1.3473, 1.5804, 1.4997,
  1.4856, 2.4087, 3.1269, 1.6846, 1.9681, 1.9013,
  2.0063, 2.2429, 4.4504, 2.5306, 2.2884, 2.2558,
  2.4650, 5.0161, 2.9737, 3.2955, 3.0433, 4.9760,
  3.3483, 3.2475, 3.6282, 4.9522, 4.0758, 3.5796,
  6.0984, 3.8736, 6.0592, 5.8478, 3.8723, 4.3039,
  6.7435, 4.4962, 5.4081, 6.5302, 4.8733, 6.5704,
  4.842, 7.589, 5.2344, 7.3317
)
knn_time <- c(
  04.2682, 04.5013, 06.571, 05.8672, 07.9289, 07.6253,
  09.5135, 09.1288, 10.3448, 11.729, 12.4124, 13.1689,
  13.8796, 14.3912, 15.3656, 16.0299, 16.553, 18.6206,
  18.5721, 18.7697, 21.1235, 20.2452, 22.4805, 21.7048,
  23.9975, 24.6640, 24.1858, 25.4328, 26.8170, 27.4834,
  28.2759, 28.9812, 30.0044, 31.1699, 32.0367, 31.6555,
  32.3268, 34.8468, 33.7590, 36.4634, 35.0449, 37.6300,
  38.1448, 37.1710, 39.8298, 40.4879, 41.4895, 42.0812,
  42.6466, 45.1496, 44.1108, 45.1389, 46.4636, 48.1075,
  47.9914, 48.4514, 48.4674, 52.1068
)

gnb_time <- c(
  0.4754, 0.4584, 0.4665, 0.4664, 0.4654, 0.4790,
  0.4640, 0.4882, 0.4865, 0.4978, 0.4741, 0.4852,
  0.5011, 0.5150, 0.4909, 0.5239, 0.5415, 0.6179,
  0.6066, 0.6655, 0.6288, 0.6132, 0.6739, 0.5548,
  0.5001, 0.5469, 0.5214, 0.5495, 0.5324, 0.5346,
  0.5238, 0.5400, 0.6438, 0.6231, 0.6194, 0.5533,
  0.5589, 0.5506, 0.5817, 0.6002, 0.7572, 0.7367,
  0.6967, 0.8846, 0.9222, 0.8777, 0.8678, 0.7493,
  0.6440, 0.6322, 0.6534, 0.6491, 0.6588, 0.6680,
  0.6659, 0.6648, 0.6706, 0.6794
)
```

```
lda_time <- c(
  0.1862, 1.6473, 0.1882, 0.2244, 0.223, 0.2709,
  0.2105, 0.2381, 0.2632, 0.2726, 0.2896, 0.3073,
  0.3339, 0.3588, 0.3524, 0.4168, 0.4302, 0.4273,
  0.4271, 0.5049, 0.4700, 0.5366, 0.5157, 0.5852,
  0.5694, 0.5962, 1.1273, 1.8477, 2.1616, 0.6892,
  0.6794, 0.7336, 0.7524, 0.7978, 0.8319, 0.7925,
  0.8841, 0.8601, 0.9127, 0.9183, 2.5774, 2.4831,
  1.0113, 1.0661, 1.1097, 1.1000, 1.1252, 1.1841,
  1.2263, 1.3570, 3.9993, 1.4007, 1.3090, 1.3469,
  1.4059, 1.4164, 1.4388, 1.6564
)

todos_dados <- data.frame(
  "bloco" = rep(blocos, 4),
  "score" = c(lr_score, knn_score, gnb_score, lda_score),
  "tempo" = c(lr_tempo, knn_time, gnb_time, lda_time),
  "metodo" = c(
      rep("LR", 58),
      rep("KNN", 58),
      rep("GNB", 58),
      rep("LDA", 58)
  )
)

#todos_dados
head(todos_dados)
```

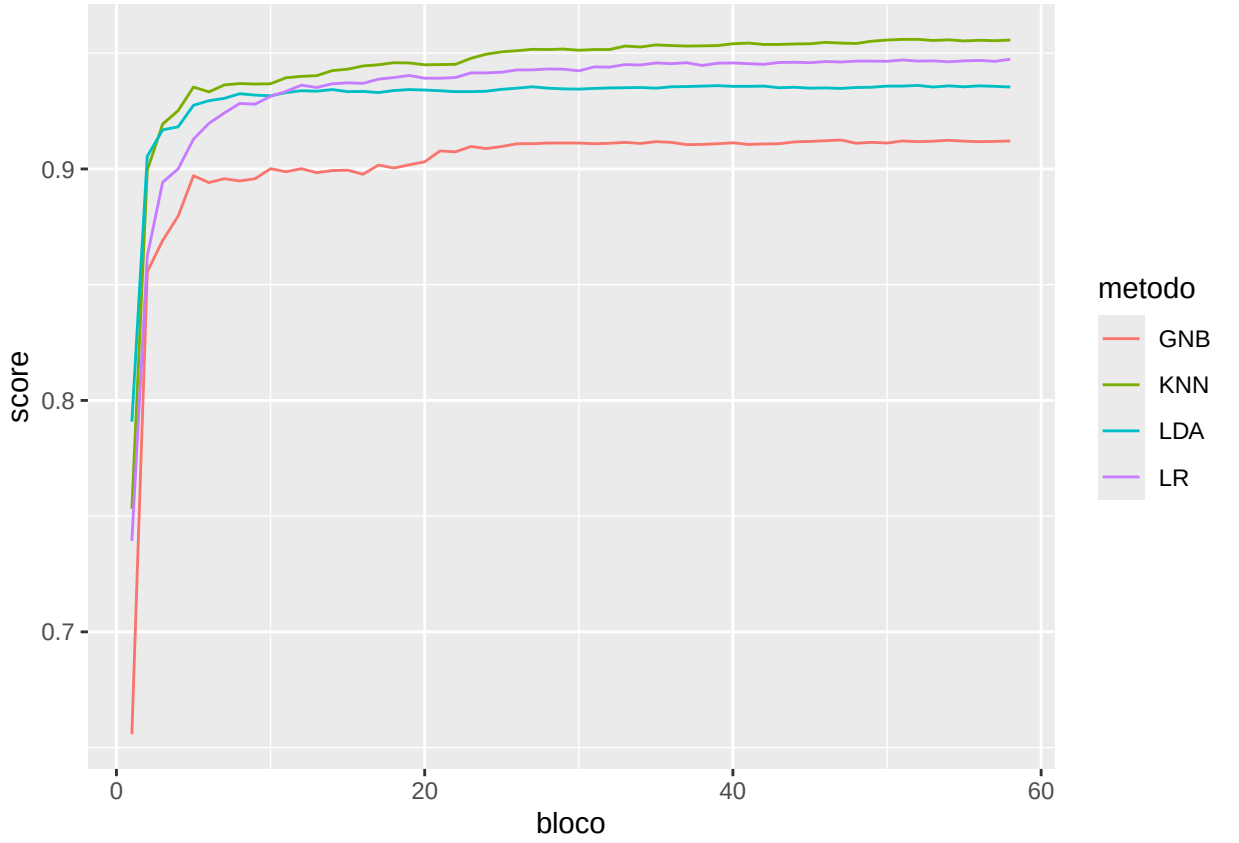
```
##   bloco  score  tempo metodo
## 1     1 0.7393 0.1580     LR
## 2     2 0.8624 0.2674     LR
## 3     3 0.8943 0.5048     LR
## 4     4 0.9000 0.3970     LR
## 5     5 0.9129 0.3819     LR
## 6     6 0.9197 0.4371     LR
```

Visualizando em gráficos

Bloco x Score

```
grafico1 <- ggplot(todos_dados, aes(x=bloco, y=score, color=metodo)) +
    geom_line(aes(group=metodo))

grafico1
```




```
ggsave("grafico1.png", grafico1)
```

```
## Saving 6.5 x 4.5 in image
```

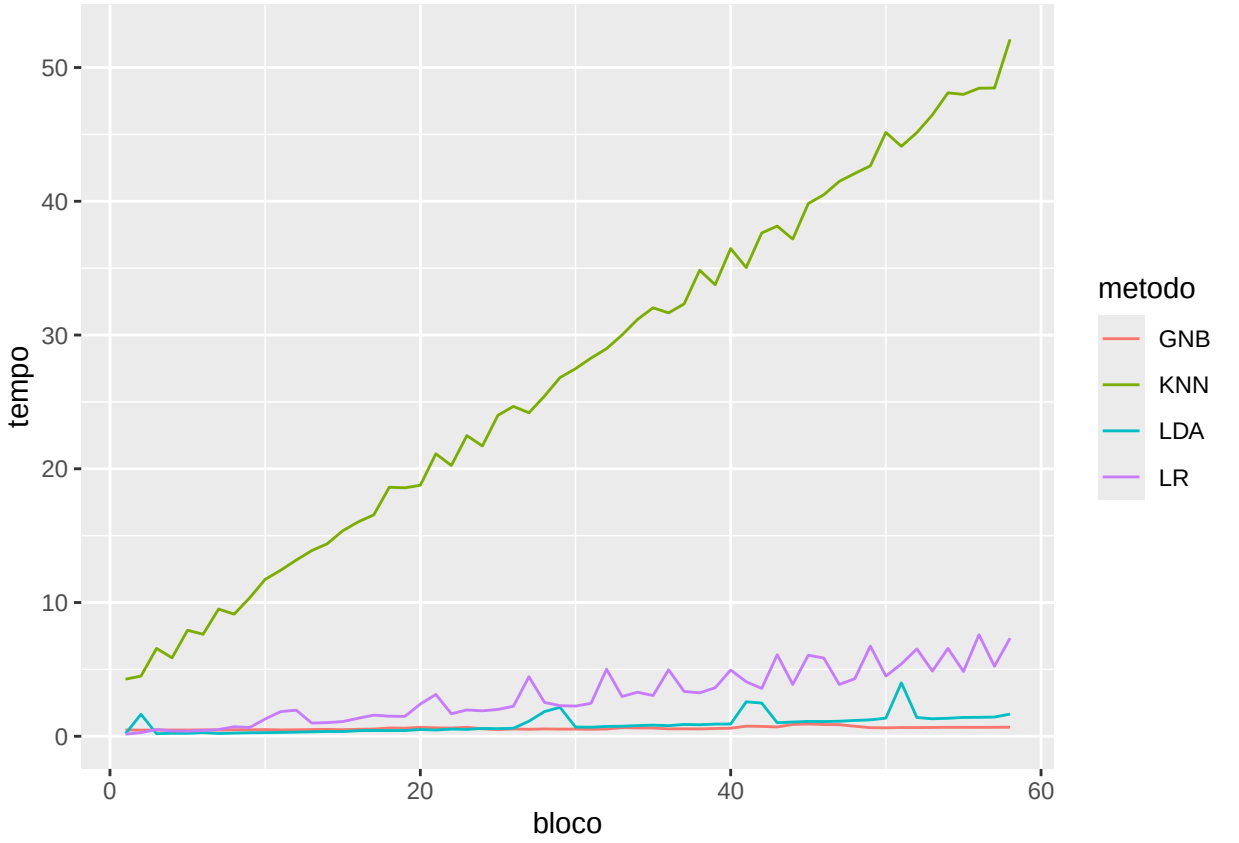
A partir desse gráfico pode se perceber que:

- o KNN teve o melhor desempenho (em score) em praticamente todos os blocos (nos blocos iniciais ficou bem semelhante com a LDA);
- o LDA tinha um melhor desempenho (em score) que a LR nos cerca de ~10 primeiros blocos, invertendo o desempenho a seguir
- a GNB teve o pior desempenho (em score) em todos os blocos
- após aproximadamente ~25 blocos todos os métodos "estabilizaram" seus scores.

Bloco x Tempo

```
grafico2 <- ggplot(todos_dados, aes(x=bloco, y=tempo, color=metodo)) +
    geom_line(aes(group=metodo))

grafico2
```




```
ggsave("grafico2.png", grafico2)
```

```
## Saving 6.5 x 4.5 in image
```

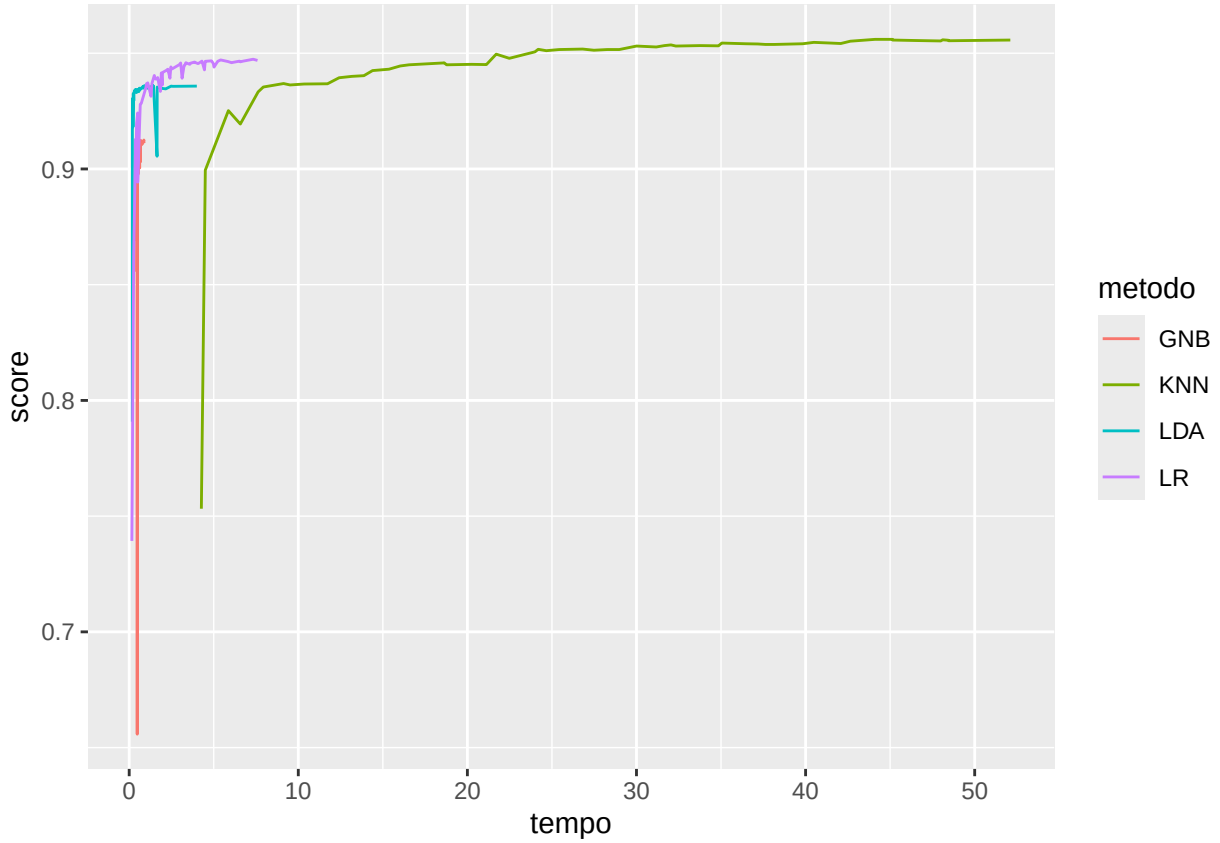
A partir desse gráfico pode se perceber que:

- o KNN foi o mais demorado em tempo de processamento em todos os blocos
- o LR teve o segundo pior tempo, exceto nos cerca de ~10 primeiros blocos.
- a LDA e a LR tiveram tempos similares, exceto picos ocasionais na LDA (blocos por volta de ~28, ~42 e ~51)

Tempo x Score

```
grafico3 <- ggplot(todos_dados, aes(x=tempo, y=score, color=metodo)) +
    geom_line(aes(group=metodo))

grafico3
```




```
ggsave("grafico3.png", grafico3)
```

```
## Saving 6.5 x 4.5 in image
```

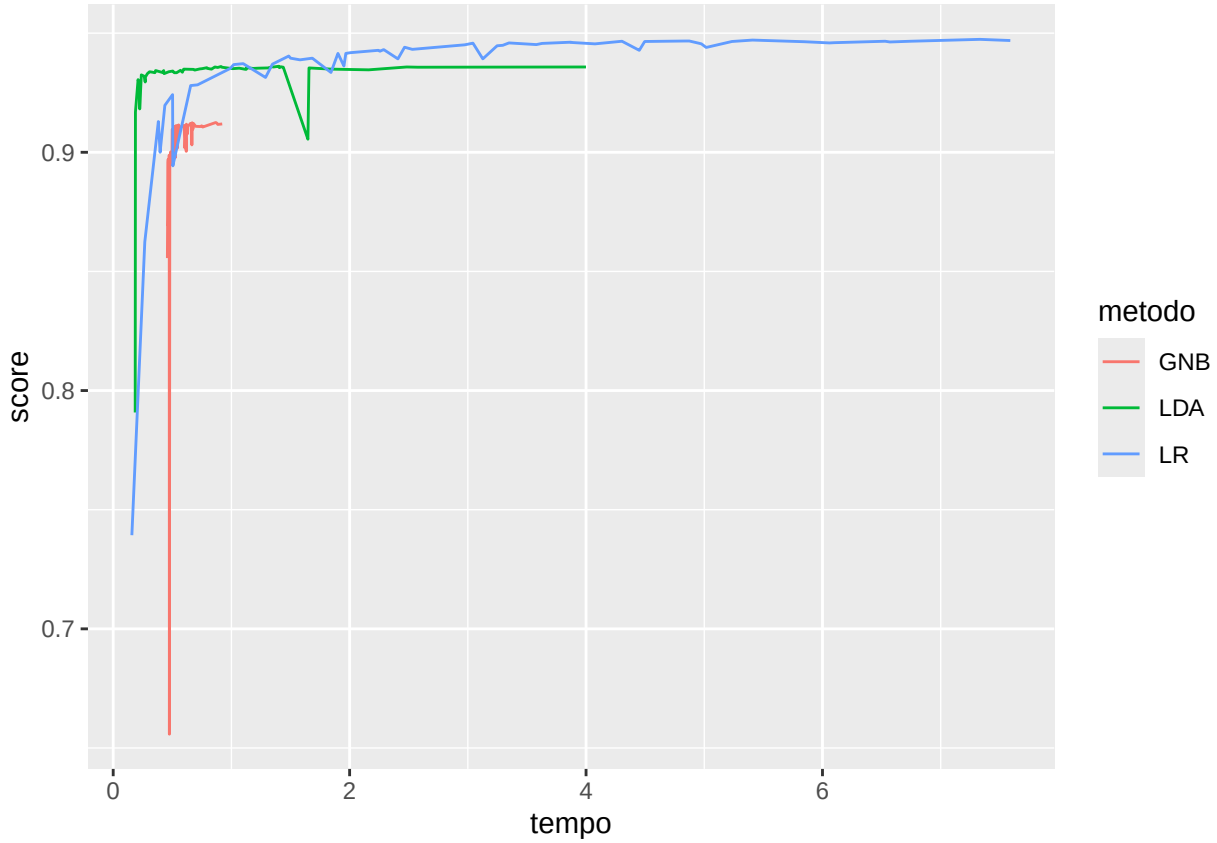
A partir desse gráfico pode se perceber que:

- a KNN, como já constatado anteriormente, foi o mais demorado, nesse caso o que demorou mais para obter o mesmo escore dos demais.
- a GNB, LDA e LR tiveram aparentemente um desempenho de tempo semelhante para tempos semelhantes, porém essa pode ser uma ilusão causada pela escala de tempo. Assim sendo fizemos um gráfico a seguir sem o KNN

Tempo x Score (sem KNN)

```
grafico4 <- todos_dados %>%
  group_by(metodo) %>%
  filter(metodo != "KNN") %>%
  ggplot(aes(x=tempo, y=score, group = metodo ,color=metodo)) +
    geom_line(aes(group=metodo))

grafico4
```




```
ggsave("grafico4.png", grafico4)
```

```
## Saving 6.5 x 4.5 in image
```

A partir desse gráfico pode se perceber que:

- o LDA foi quem teve o melhor desempenho (score) considerando um tempo menor que aproximadamente 1,5. Após esse tempo a LER passou a ter melhor desempenho (score) proporcional ao tempo usado.
- Avaliando apenas esses 3 métodos o GNB teve o pior desempenho (score) para um mesmo tempo.

3D - não funcionou

```
## Tentei fazer um grafico 3d mas não deu certo, não tive tempo o suficiente para tentar de outras forma

#grafico5 <- plot_ly(todos_dados, x= ~bloco, y=~score, z =~tempo) %>% add_surface() %>% layout()
#grafico5
```

Questinomaentos e Respostas

1. Qual é o classificador que tem o melhor desempenho com poucos dados $<$ 1000 exemplos.

Com 1000 dados foi nossa primeira tentativa nos modelos, de acordo com os nossos dados (a seguir) o melhor score no primeiro bloco foi obtido pelo método LDA.

```
bloco1 <- todos_dados %>%
  group_by(metodo) %>%
  filter(bloco == 1)

bloco1
```

```
## # A tibble: 4 x 4
## # Groups:   metodo [4]
##   bloco score tempo metodo
##   <int> <dbl> <dbl> <chr>
## 1     1 0.739 0.158 LR
## 2     1 0.753 4.27  KNN
## 3     1 0.656 0.475 GNB
## 4     1 0.791 0.186 LDA
```

```
# equivale a...
# bloco1 <- group_by(.data = todos_dados, metodo)
# bloco1 <- filter(.data = bloco1, bloco == 1)
#
# bloco1
```

2. Qual é o classificador que tem melhor desempenho com todos os dados.

De acordo com os nossos dados (a seguir) o melhor score no 58º bloco (todos os dados) foi obtido pelo método KNN, embora com um gasto de tempo bem maior que os demais.

```
bloco58 <- todos_dados %>%
  group_by(metodo) %>%
  filter(bloco == 58)

bloco58
```

```
## # A tibble: 4 x 4
## # Groups:   metodo [4]
##   bloco score  tempo metodo
##   <int> <dbl>  <dbl> <chr>
## 1    58 0.947  7.33  LR
## 2    58 0.956 52.1   KNN
## 3    58 0.912  0.679 GNB
## 4    58 0.935  1.66  LDA
```

3. Qual é o classificador é mais rápido para classificar os 58k exemplos de teste.

De acordo com os nossos dados (a seguir) o melhor tempo no 58º bloco foi obtido pelo método GNB, embora ele tenha tido o pior desempenho (em score) para esse bloco.

```
bloco58 <- todos_dados %>%
  group_by(metodo) %>%
  filter(bloco == 58)

bloco58
```

```
## # A tibble: 4 x 4
## # Groups:   metodo [4]
##   bloco score  tempo metodo
##   <int> <dbl>  <dbl> <chr>
## 1    58 0.947  7.33  LR
## 2    58 0.956 52.1   KNN
## 3    58 0.912  0.679 GNB
## 4    58 0.935  1.66  LDA
```

O que você pode dizer a respeito das matrizes de confusão. Os erros são os mesmos para todos os classificadores quando todos eles utlizam toda a base de treinamento?

Os erros não são os mesmos para todos os modelos, ou seja, cada um apresenta padrões diferentes de acertos e erros (cujo desempenho vai refletir em parte no score obtido). Podemos observar que: - o KNN parece ter o melhor desempenho, especialmente em classes 8 e 9. - o LDA parece ter um bom desempenho com poucos erros razoavelmente equilibrados entre as classes. - o LR teve um bom desempenho geral, mas muitos erros nas classes 8 e 9. - o GaussianNB teve o pior desempenho, com dificuldades especialmente nas classes 0 e 8.

Codigo

```
#!/usr/bin/python
# -*- encoding: iso-8859-1 -*-

import sys
import numpy as np
import time

from sklearn import linear_model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_svmlight_file

from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import Perceptron

def main(X_train, y_train, X_test, y_test, history):
    # cria o classificador - ESCOLHER UM
    clf = linear_model.LogisticRegression()
    #clf = KNeighborsClassifier(n_neighbors=3, metric=euclidean)
    #clf = GaussianNB()
    #clf = LinearDiscriminantAnalysis()

    X_train_dense = X_train.toarray()
    clf.fit(X_train_dense, y_train)

    X_test_dense = X_test.toarray()
    y_pred = clf.predict(X_test_dense)
```

```
    # mostra o resultado do classificador na base de teste
    history.append( "{:5.4f}".format(clf.score(X_test_dense, y_test)))
    print("CLF score = ", "{:5.4f}".format(clf.score(X_test_dense, y_test)))

    # cria a matriz de confusao
    #cm = confusion_matrix(y_test, y_pred)
    #print(cm)

    #print(y_predProb)

if __name__ == "__main__":
    #if len(sys.argv) != 3:
    #   sys.exit("Use: lab2.py <dataTR> <dataTS>")
    arq_test = "test.txt"
    arq_train = "train.txt"

    # loads data
    print ("Loading data...")
    X_train, y_train = load_svmlight_file(arq_test)
    X_test, y_test = load_svmlight_file(arq_train)
    size = X_train.shape
    batchsize = 1000
    ini = 0
    end = batchsize

    history = []
    tempo = []

    while(end <= size[0] ):
        xt = X_train[0:end]
        yt = y_train[0:end]
        print ("Training size... ", end)
        start_time = time.time()
        main(xt, yt, X_test, y_test, history)
        tempo.append(round((time.time() - start_time), 4))
        print("--- %6.8f seconds ---" % (time.time() - start_time))
        end = end + batchsize

    print("Scores")
    print(history)

    print("Tempo")
    print(tempo)
```