Reconhecimento Ótico de Caracteres usando SVM

Heitor Gabriel S. Monteiro

22/10/2021

Contents

1 Dados

```
setwd('/home/heitor/Área de Trabalho/R Projects/Análise Macro/Lab 10')

library(tidyverse)
library(tidymodels)
library(workflowsets)
library(kernlab)
library(kableExtra)
```

```
library(ggside)
library(plotly)
library(gridExtra)
```

Os dados são estatísticas de uso de *piexels* por letras do alfabeto com diferentes estilos.

Figure 1: Exemplos de tipos de letras

Ao exportar os dados e ter uma visão geral sobre as variáveis envolvidas, transformamos as letras em fatores:

```
dt <- read_csv("letterdata.csv") %>%
    as_tibble()

dt %>% glimpse()
```

```
## Rows: 20,000
## Columns: 17
## $ letter <chr> "T", "I", "D", "N", "G", "S", "B", "A", "J", "M", "X", "O", "G"~
## $ xbox   <dbl> 2, 5, 4, 7, 2, 4, 4, 1, 2, 11, 3, 6, 4, 6, 5, 6, 3, 7, 6, 2, 1,~
```

```
## $ ybox   <dbl> 8, 12, 11, 11, 1, 11, 2, 1, 2, 15, 9, 13, 9, 9, 9, 9, 4, 10, 11~
## $ width  <dbl> 3, 3, 6, 6, 3, 5, 5, 3, 4, 13, 5, 4, 6, 8, 5, 5, 4, 5, 6, 3, 2,~
## $ height <dbl> 5, 7, 8, 6, 1, 8, 4, 2, 4, 9, 7, 7, 7, 6, 7, 4, 3, 5, 8, 3, 2, ~
## $ onpix  <dbl> 1, 2, 6, 3, 1, 3, 4, 1, 2, 7, 4, 4, 6, 9, 6, 3, 2, 2, 5, 1, 1, ~
## $ xbar   <dbl> 8, 10, 10, 5, 8, 8, 8, 8, 10, 13, 8, 6, 7, 7, 6, 10, 8, 6, 6, 1~
## $ ybar   <dbl> 13, 5, 6, 9, 6, 8, 7, 2, 6, 2, 7, 7, 8, 8, 11, 6, 7, 8, 11, 6, ~
## $ x2bar  <dbl> 0, 5, 2, 4, 6, 6, 6, 2, 2, 6, 3, 6, 6, 6, 7, 3, 7, 6, 5, 3, 2, ~
## $ y2bar  <dbl> 6, 4, 6, 6, 6, 9, 6, 2, 6, 2, 8, 3, 2, 5, 3, 5, 5, 8, 6, 6, 5, ~
## $ xybar  <dbl> 6, 13, 10, 4, 6, 5, 7, 8, 12, 12, 5, 10, 6, 7, 7, 10, 7, 11, 11~
## $ x2ybar <dbl> 10, 3, 3, 4, 5, 6, 6, 2, 4, 1, 6, 7, 5, 5, 3, 5, 6, 7, 9, 4, 5,~
## $ xy2bar <dbl> 8, 9, 7, 10, 9, 6, 6, 8, 8, 9, 8, 9, 11, 8, 9, 7, 8, 11, 4, 9, ~
## $ xedge  <dbl> 0, 2, 3, 6, 1, 0, 2, 1, 1, 8, 2, 5, 4, 8, 2, 3, 2, 2, 3, 0, 0, ~
## $ xedgey <dbl> 8, 8, 7, 10, 7, 8, 8, 6, 6, 1, 8, 9, 8, 9, 7, 9, 8, 8, 12, 7, 7~
## $ yedge  <dbl> 0, 4, 3, 2, 5, 9, 7, 2, 1, 1, 6, 5, 7, 8, 5, 6, 3, 5, 2, 1, 0, ~
## $ yedgex <dbl> 8, 10, 9, 8, 10, 7, 10, 7, 7, 8, 7, 8, 8, 6, 11, 9, 8, 9, 4, 7,~
```

```
dt <- dt %>% mutate(letter = factor(letter))
dt$letter %>% levels()
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
dt %>% summary()
```

```
##      letter           xbox             ybox            width       
##  U      :  813   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
##  D      :  805   1st Qu.: 3.000   1st Qu.: 5.000   1st Qu.: 4.000  
##  P      :  803   Median : 4.000   Median : 7.000   Median : 5.000  
##  T      :  796   Mean   : 4.024   Mean   : 7.035   Mean   : 5.122  
##  M      :  792   3rd Qu.: 5.000   3rd Qu.: 9.000   3rd Qu.: 6.000  
##  A      :  789   Max.   :15.000   Max.   :15.000   Max.   :15.000  
##  (Other):15202                                                     
##      height           onpix            xbar            ybar       
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.0  
##  1st Qu.: 4.000   1st Qu.: 2.000   1st Qu.: 6.000   1st Qu.: 6.0  
##  Median : 6.000   Median : 3.000   Median : 7.000   Median : 7.0  
##  Mean   : 5.372   Mean   : 3.506   Mean   : 6.898   Mean   : 7.5  
##  3rd Qu.: 7.000   3rd Qu.: 5.000   3rd Qu.: 8.000   3rd Qu.: 9.0  
##  Max.   :15.000   Max.   :15.000   Max.   :15.000   Max.   :15.0  
##                                                                   
##      x2bar            y2bar            xybar           x2ybar       
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
##  1st Qu.: 3.000   1st Qu.: 4.000   1st Qu.: 7.000   1st Qu.: 5.000  
```

```
##  Median : 4.000   Median : 5.000   Median : 8.000   Median : 6.000
##  Mean   : 4.629   Mean   : 5.179   Mean   : 8.282   Mean   : 6.454
##  3rd Qu.: 6.000   3rd Qu.: 7.000   3rd Qu.:10.000   3rd Qu.: 8.000
##  Max.   :15.000   Max.   :15.000   Max.   :15.000   Max.   :15.000
##
##      xy2bar           xedge           xedgey           yedge
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 7.000   1st Qu.: 1.000   1st Qu.: 8.000   1st Qu.: 2.000
##  Median : 8.000   Median : 3.000   Median : 8.000   Median : 3.000
##  Mean   : 7.929   Mean   : 3.046   Mean   : 8.339   Mean   : 3.692
##  3rd Qu.: 9.000   3rd Qu.: 4.000   3rd Qu.: 9.000   3rd Qu.: 5.000
##  Max.   :15.000   Max.   :15.000   Max.   :15.000   Max.   :15.000
##
##      yedgex
##  Min.   : 0.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 7.801
##  3rd Qu.: 9.000
##  Max.   :15.000
##
```

Vemos as médias e desvios-padrão da quantidade de pixels usados na imagem, média de piexels por linha e por coluna.

```
dt1 <- dt %>%
    group_by(letter) %>%
    summarise( Média_pix    = mean(onpix),
               Var_pix      = sd(onpix)  ,
               Média_Linha  = mean(xbar),
               Var_Linha    = sd(xbar)  ,
               Média_Col    = mean(ybar),
               Var_Col      = sd(ybar)  )
dt1 %>%
    kable(#format = 'html',
          align = 'c',
          caption = 'Estatísticas Descritivas dos Pixels das Letras') %>%
    kable_styling(full_width = F,
                  bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

```
gg1 <- dt %>%
    ggplot( )+
    geom_boxplot(aes(y=onpix,
```

Table 1: Estatísticas Descritivas dos Pixels das Letras

letter	Média_pix	Var_pix	Média_Linha	Var_Linha	Média_Col	Var_Col
A	2.991128	1.784357	8.851711	1.9588567	3.631179	1.8611964
B	4.596606	2.209413	7.671018	1.1997258	7.062663	0.8409395
C	2.775815	1.708725	5.437500	1.1588905	7.627717	0.9418426
D	4.026087	1.985160	7.539130	1.6322380	6.806211	0.9514122
E	3.679688	1.897368	5.966146	1.9293654	7.352865	0.8661185
F	3.178065	2.058394	4.913548	2.5601820	10.454194	1.8629128
G	3.566623	2.056148	6.866753	1.0034110	6.586029	1.0758351
H	4.253406	2.218263	7.344687	1.1713929	7.320163	1.1502189
I	1.825166	1.717796	7.458278	1.0982478	7.035762	0.9412690
J	2.315930	1.645304	9.665328	2.1889106	5.666667	2.1095976
K	3.981056	2.148213	5.592693	1.9029290	7.070365	0.9079154
L	2.649146	1.826458	4.800263	2.8995553	3.592641	1.9357165
M	5.267677	2.728612	7.641414	1.6933579	6.407828	1.4281226
N	3.564496	1.857622	7.012771	1.4665374	7.952746	1.1450168
O	3.503320	1.909607	7.341302	0.7712293	6.965471	0.9655656
P	3.735990	2.220969	6.219178	1.6642885	9.955168	1.8605988
Q	4.136654	2.118720	8.160920	0.9593557	6.808429	1.8475339
R	4.187335	2.097806	7.147757	1.5219448	8.122691	1.5071778
S	3.486631	2.057040	7.811497	1.0338457	6.945187	1.4711654
T	2.858040	1.929238	6.428392	1.3179546	11.369347	2.2164088
U	3.325953	2.126140	6.116851	1.6815059	6.936039	1.6206313
V	2.815445	2.017284	6.056283	2.0672126	10.136126	1.8782898
W	4.851064	2.643314	6.078457	2.3108529	9.214096	1.5908069
X	3.213469	1.820207	7.252859	1.2942823	7.171538	0.7078643
Y	3.057252	2.437610	6.436387	2.0974110	9.496183	1.6946858
Z	3.250681	1.681240	7.525886	1.1744252	7.125341	1.2861253

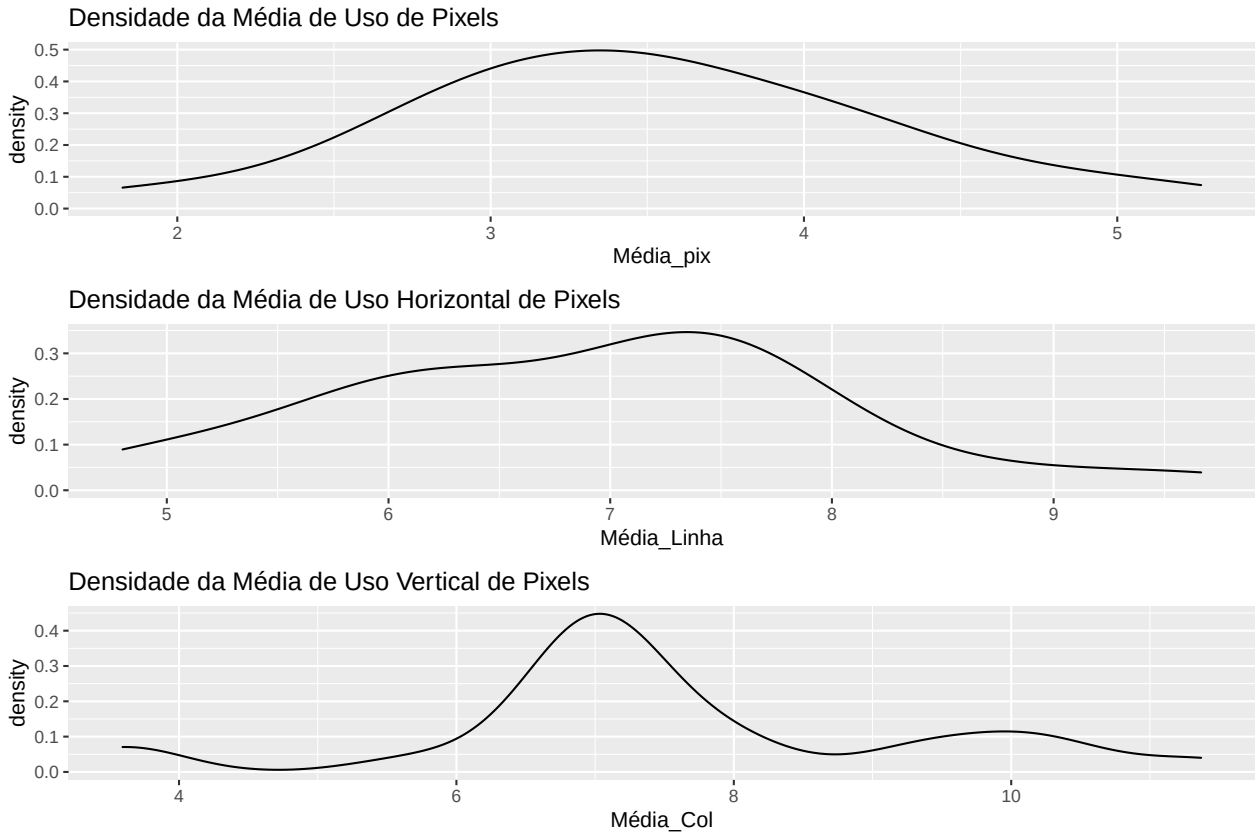
```
                    x=letter,
                    color=letter))+
    theme(legend.position = "none")+
    labs(title = 'Box-Plot do Uso de Pixels por Letras') +
    ylab('Porcentagem de Pixels')

ggplotly(gg1)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is i
```

Box-Plot do Uso de Pixels por Letras

```
gg2 <- dt1 %>%
    ggplot() +
    geom_density(aes(Média_pix)) +
    labs(title = 'Densidade da Média de Uso de Pixels')
gg3 <- dt1 %>%
    ggplot() +
    geom_density(aes(Média_Linha))+
    labs(title = 'Densidade da Média de Uso Horizontal de Pixels')
gg4 <- dt1 %>%
    ggplot() +
    geom_density(aes(Média_Col))+
    labs(title = 'Densidade da Média de Uso Vertical de Pixels')
grid.arrange(gg2, gg3, gg4)
```



Concluímos que as letras têm grupos de médias, com alta variabilidade entre elas. A variabilidade e a assimetria da distribuição podem ser espaços vetoriais adicionados para a de análise, corroborando o uso de um Kernel Linear. Ainda sim, como nosso propósido é classificar o caractere, treinaremos o modelo com vários Kernels e veremos qual se encaixa melhor no teste.

2 Separação dos dados:

```
slice_1 <- initial_split(dt)
train   <- training(slice_1)
test    <- testing(slice_1)
```

3 Modelo

Vamos criar a estrutura geral do nosso modelo, deixando espaços livres com `tune()` por serem os parâmetros a serem testados com vários kernels e vários parâmetros de custo, reiteragas vezes.

```
rbf_svm_algort <- svm_rbf(cost = tune(),
                          rbf_sigma = tune()) %>%
    set_engine("kernlab") %>%
    set_mode("classification")
```

4 Tratamentos e Fórmula para Alimentar o Modelo

Defino como os dados alimentarão o modelo já descrito acima e aplico um tratamento de normalização nos dados, usando desvio da média e desvio-padrão.

```
recipe_svm <-
    recipe(letter ~ .,
           data = train) %>%
    step_normalize(all_numeric_predictors()) %>%
    prep()
```

5 Workflow

Junto o modelo descrito e os dados tratados, formando um workflow:

```
wrkflw_1 <- workflow() %>%
    add_model(rbf_svm_algort) %>%
    add_recipe(recipe_svm)
```

6 Validação Cruzada

Defino a validação cruzada em grupos de cinco, ou seja, a amostra de treino será $\frac{4}{5}$ passando por várias reamostragens.

```
valid_1 <- vfold_cv(train, v = 5)
```

7 Treinamento

Como testar todas as combinações possíveis de parâmetros sobrecarregará a máquina, para fins de exercício, definirei um intervalo que o algoritmo deve procurar os melhores parâmetros:

```
start_grid_1 <-
    wrkflw_1 %>%
    parameters() %>%
    update(
        cost = cost(c(-1, 1)),
        rbf_sigma = rbf_sigma(c(-2, 2))
    ) %>%
    grid_regular(levels = 1)
```

Treinaremos o modelo com vários parâmetros e selecionaremos de acordo com `roc_auc`: a área abaixo da curva de ROC, um gráfico usado para diagnosticar modelos de classificação binária em geral. Sempre lembrando que defini um intervalo específico, então, vamos mostrar um ótimo local.

```
trained_svm_1 <-
    wrkflw_1 %>%
    tune_grid(resamples = valid_1,
              grid = start_grid_1,
              metrics = metric_set(roc_auc))

collect_metrics(trained_svm_1)
```

```
## # A tibble: 1 x 8
##    cost rbf_sigma .metric .estimator  mean     n std_err .config
##   <dbl>     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   0.5      0.01 roc_auc hand_till  0.836     5 0.00243 Preprocessor1_Model1
```

```
trained_svm_1 %>% show_best(n=15)
```

```
## # A tibble: 1 x 8
##    cost rbf_sigma .metric .estimator  mean     n std_err .config
##   <dbl>     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   0.5      0.01 roc_auc hand_till  0.836     5 0.00243 Preprocessor1_Model1
```

8 Testando

Selecionaremos o melhor modelo, usando o `roc_auc`.

```
best_tune  <- select_best(trained_svm_1,
                          'roc_auc',
                          n=1)
```

```
final_svm <- rbf_svm_algort %>%
    finalize_model(best_tune)

final_svm
```

```
## Radial Basis Function Support Vector Machine Specification (classification)
##
## Main Arguments:
##   cost = 0.5
##   rbf_sigma = 0.01
##
## Computational engine: kernlab
```

Aplicaremos esse modelo, `final_svm` na partição feita em `slice_1` e com a organização dos dados de acordo com `recipe_svm`. Vemos que conseguimos 80,78% de acurácia do modelo.

```
final_svm_wrkflw <- workflow() %>%
    add_recipe(recipe_svm) %>%
    add_model(final_svm) %>%
    last_fit(slice_1) %>%
    collect_predictions()

final_svm_wrkflw %>% count(letter==.pred_class)
```

```
## # A tibble: 2 x 2
##   `letter == .pred_class`     n
##   <lgl>                   <int>
## 1 FALSE                     860
## 2 TRUE                     4140
```