Projekt 2 - drużyna1

Uczestnicy: Bogna Pawlus

2023-01-27

Projekt 2

Wczytajmy dane odpowiednio do zmiennych tab1, tab2, . . . , tab10. Wykonując

```
summary(data1$target)
dim(data1)
```

gdzie za data1 wstawiamy odpowiednie dane, otrzymujemy, że elementy z niektórych grup z data1$target pojawiają się mało razy (najmniej->6), dlatego będziemy szacować jakość dobranych zmiennych, korzystając z walidacji krzyżowej, gdzie nfolds = 2, przy czym zbiory testowy i treningowy będą zawierać tę samą(+-1) liczbę obserwacji należących do danej grupy nfolds. Dla ulepszenia szacowania jakości modelu, wylosujemy zbiór treningowy 200 razy (1)

Na początku, żeby przyspieszyć czas działania algorytmu, zaczniemy od selekcji wstępnej 1000 cech ze wszystkich kilkunastu tysięcy dla zbiorów data1 = tab1, tab2, . . . , tab10:

```
set.seed(222)
task = as_task_classif(data1, target = "target")
filter = flt("kruskal_test")
ans = as.data.table(filter$calculate(task))

list_of_cols = c()
N = 1000
for (i in 1:N) {
  print(match(ans[i, 1],names(data1)))
  list_of_cols[i] = match(ans[i, 1],names(data1))
}
list_of_cols = sort(list_of_cols)
list_of_cols ## <- tutaj znajduje się wstępne 1000 kolumn
```

Na pozostałych 1000 cech zrobimy algorytm genetyczny. Wprowadźmy wstępną funkcję ewaluacyjną: Ze względu na małą liczbę obserwacji, żeby zredukować liczbę cech, użyjemy miary bacc - (k-2)/500

```
evalFunc <- function(x) {
  #zamieniamy 01 na numery kolumn
  start = 1
  for (i in 1:1000){
    if(x[i] == 1) {
      list_of_cols_tmp[start] = list_of_cols[i]
      start = start + 1
```

```
  }
}

##liczymy miarę bac - (k-2)/500.
## korzystamy z 2-krotnej walidacji krzyżowej
if (length(list_of_cols_tmp) > 100 || length(list_of_cols_tmp) == 1) {
  return(0)
}
else {
  jakosc = c() #tutaj będą wyniki dla każdej z 2 części cv

  ##podzielimy data1 na treningowy i testowy, tak żeby każdy z poziomów
  ##z data1$target występował w danym zbiorze tę samą liczbę razy
  lev = levels(data1$target)
  target = data1$target
  podzial = rep(0, length(target)) ##zaznaczamy co jest w zbiorze treningowym a co w testowym

  for(i in 1:length(lev)){
    ind_of_level = which(target == lev[i])
    podzial_lev = cut(1:length(ind_of_level), 2, labels=FALSE)
    podzial_lev = sample(podzial_lev)

    for(i in 1:length(podzial)) {
      podzial[ind_of_level[i]] = podzial_lev[i]
    }
  }

  feats = list_of_cols_tmp #te cechy (z chromosomu) wybrano i użyto w tej funkcji.

  for (i in 1:2) {
    train_ind = which(podzial != i)
    dt_tr = data1[train_ind, ]
    model <- e1071::svm(dt_tr[, feats, with = FALSE], dt_tr[, factor(target)],
                        type = "C-classification", kernel = "linear", cost = 1,
                        gamma = 1/length(feats), scale = TRUE)
    ## liczenie predykcji
    walid = data1[!train_ind, ..feats]
    numb = ncol(data1)
    walid_ans = data1[!train_ind, ..numb] #poprawna odpowiedz
    pred = predict(model, walid)
    pred = data.frame(pred)
    pred$prawdziwe = walid_ans
    pred = as.data.table(pred)
    cm = confusionMatrix(pred$prawdziwe, pred$pred)

    #liczymy jakość predykcji (bacc):
    if (ncol(cm$table) > 1) {
      cm = cm$table
      bacc = c()
      for (ii in 1:ncol(cm)) {
        bacc[ii] = cm[ii, ii]/(rowSums(cm)[ii])
      }
      jakosc[i] = mean(bacc)
```

```
      }
    }
    #zwracamy z minusem, bo rbga.bin optymalizuje minimum
    return(-(mean(jakosc) - (length(list_of_cols_tmp) - 2) / 500))
  }
}
```

Teraz wprowadzamy główną funkcję ewaluacyjną, która polepszy miarę bacc (wniosek (1)):

```
evalFuncBetter <- function(x) {
  answers = c()
  for (s in 1:200) {
    answers[s] = evalFunc(x)
  }
  return(mean(answers))
}
```

Funkcja zamieniająca chromosom z 0 i 1 na numery kolumn:

```
change01to_numbers <- function(x) {
  start = 1
  for (i in 1:1000){
    if(x[i] == 1) {
      cat(list_of_cols[i])
      if (start < leng) {
        cat(",")
      }
      start = start + 1
    }
  }
}
```

Na końcu dla wszystkich zbiorów data1 = tab1, tab2, . . . , tab10 przeprowadzimy algorytm genetyczny:

```
iter = 20
GAmodel <- rbga.bin(size = 1000, popSize = 20, iters = iter, mutationChance = 0.004,
                    elitism = T, evalFunc = evalFunc, zeroToOneRatio=10)
```

Po wykonaniu

```
bestSolution<-GAmodel$population[which.min(GAmodel$evaluations),]
leng = sum(bestSolution)
change01to_numbers(bestSolution)
```

otrzymamy numery kolumn ze zgłoszenia.

Wykonując

```
min(GAmodel$best)
```

otrzymujemy, że miara bac - (k-2)/500 dla wybranych cech w modelach tab1, tab2, . . . , tab10 to odpowiednio:

```
## [1] "[1] -0.7790833 -0.7284524 -0.7053681 -0.8406439 -0.7911443"
```

```
## [1] "[6] -0.7378493 -0.8555833 -0.6122578 -0.8052463 -0.4911713"
```

czyli średnia miara to

```
## [1] -0.73468
```

Ostateczne wypisane zgłoszenie:

```
##  [1]   343   648  3750  3932  4128  5027  5896  6093  7134  8337  9157 11259
## [13] 11468 12469 12473 13199 13790 14669 15817 16260 16823 16855 17843 18833
## [25] 19483 19837 20389 21089 21291 23650 24527 25047 25229 26478 26525 27468
## [37] 27884 28360 29082 29361 30294 34395 35406 37579 40432 40669 41387 41742
## [49] 41992 42158 42378 42500 43995 44383 44402 44915 44945 45121 45297 46987
## [61] 47089 49647 50596 50772 50911 52828 54370 55444 58327 58338 58489 59557
## [73] 59595 60343 60651 61081
```

```
##  [1]   324   900  1206  1639  2473  2657  2921  2996  3016  3050  3275  3514
## [13]  3751  4086  4230  4284  4810  5032  5201  5302  5834  5976  6936  7518
## [25]  8680  8962  9000  9120  9153  9235  9735  9988 10931 11658 11735 12048
## [37] 12094 15302 16523 16808 17993 18018 18054 18282 18589 18638 19592 19612
## [49] 20205 20548 20863 21145 21333 21372 21435 22905 24089 24211 25306 25859
## [61] 26307 27681 28640 31444 34202 37871 38835 40166 42635 47820 52307 53054
```

```
##  [1]   656   859   937   947  1136  1505  1937  2503  2627  2863  2952  3031
## [13]  3086  3387  3617  3643  3962  4487  4721  4760  5074  5343  5649  5775
## [25]  6158  6453  8663  8890  9475  9644  9974 10050 10076 10581 11179 12389
## [37] 12667 13012 13130 14543 16869 17802 18459 18715 18928 18942 19113 19503
## [49] 21561 21581 21592 21611 22264 22371 23328 23753 24376 24782 24867 25187
## [61] 25738 26429 26458 26503 26724 26950 27067 27533 27650 28576 28778 29039
## [73] 29350 29607 30497 30728 34833 37226 37821 39471 43052 46037 47703 49261
```

```
##  [1]   521   840  1453  1765  2043  2162  2264  2302  2629  2755  2759  2982
## [13]  3242  3311  3530  3853  3978  4707  4864  4980  5107  5185  5533  6561
## [25]  6918  8926  8973  8976  9097  9465  9664  9964 10228 10245 10566 10828
## [37] 10843 11025 11276 11913 12015 12090 12784 13147 13292 13324 13665 13817
## [49] 13819 13914 14068 14178 14281 14344 14643 15659 17188 18000 18599 18668
## [61] 18846 18959 19209 19583 19874 20558 21440
```

```
##  [1]   260   617  1208  1437  1879  2010  2126  2415  2727  2842  4104  4365
## [13]  4734  4743  4747  4983  5297  5412  6088  6119  7625  7788  8014  8060
## [25]  8977  9165 10057 11900 11946 12386 13293 13371 13416 13505 14082 14368
## [37] 15287 15714 16269 16395 17117 19014 19088 20884 21427 22218 23240 23454
## [49] 23583 24511 24870 24969 26254 26328 27106 28103 28569 29931 30636 31983
## [61] 32031 34213 35371 35712 35920 36396 38581 38904 44936 48439 49279 50239
```

```
##  [1]   516  1049  1185  1265  1456  1541  1622  1711  1788  2154  2166  2170
## [13]  3023  3210  3553  3601  3648  3897  4429  4532  6539  6853  7362  7366
## [25]  7676  8691  8839  8950  8962  9013  9032  9187  9714 10232 10549 10717
## [37] 10732 11541 11676 11703 11993 12090 12189 12201 12224 12268 12327 12548
## [49] 12637 12692 13724 14617 15148 15410 15830 17659 17736 17811 17815 17818
## [61] 18184 18856 19241 19246 19380 19991 20234 21491
```

```
##  [1]   263   842   997  1584  1623  1859  1906  2269  2398  2718  3310  3327
## [13]  3518  4223  4277  4586  5320  5584  7012  7191  8362  8368  8411  8826
## [25]  9816 10051 10232 10610 10852 11037 11337 11665 11925 12146 12380 12381
## [37] 12505 12846 12914 13000 13611 13736 13851 14397 14696 14834 14853 15354
## [49] 15474 15484 15854 16017 16086 16228 16264 16466 17111 18716 18984 19048
## [61] 19067 19522 20098 20168 21093 21305 21658 22011 22188
```

```
##  [1]   105   986  2175  2288  2731  3157  6158  7166  9130 10422 10481 10853
## [13] 10922 11932 12657 16028 16185 17052 17255 17322 18308 18992 19251 19786
## [25] 20014 21082 21864 21962 24458 25209 25384 25469 26170 26459 28516 28997
## [37] 29435 29469 30431 31986 34064 34201 34338 35907 36435 36724 37206 37899
## [49] 39275 39373 42701 42830 44048 44295 45637 46046 46456 47165 47695 48062
## [61] 48851 49714 54704 55370 56543 57701 58476
```

```
##  [1]   551  1536  1970  2331  2416  3476  3680  4203  4577  4765  5916  6226
## [13]  6746  7399  7870  8908  9213  9422  9832 10042 10345 10724 11589 11629
## [25] 11730 11773 11991 12047 12098 12545 12546 13346 14350 14828 15932 16378
## [37] 16905 17996 18194 18282 18660 19130 19640 20003 20762 20908 21162 21753
## [49] 22006 22092 22592 22925 23705 24315 24496 25644 25682 26044 27131 27387
## [61] 28262 29473 31219 31404 31829 34564 34639 34685 35840 36541 37171 37230
## [73] 38928 39749 40116 41142 43626 44925 51582 53849
```

```
##  [1]    12   166   677   712  1124  1599  1732  2016  2129  2259  2497  3332
## [13]  4173  4233  4411  4898  5372  5687  5741  6085  7216  8776  9314  9598
## [25] 10020 10130 10780 10977 11064 11784 11989 12171 12187 12190 12575 13404
## [37] 15871 15955 16096 17685 17729 17733 17766 17990 18158 18652 19156 19178
## [49] 19704 20512 20729 21386 22500 22536 23896 23954 24182 24290 24717 25496
## [61] 25700 27201 27670 28254 29610 30262 34175 38301 38408 38765 40133 42544
## [73] 44086 45420 48181 49009 51517 51887 54084 54096
```