Comparando listas e rankings

Fernando Náufel

11/02/2024 18:55

Índice

Apresentação

???

1 Listas e *rankings*

1.1 Problema

Vamos trabalhar com listas e *rankings* sujeitos às seguintes condições:

- A lista tem k elementos, $k > 0$, não ordenados.
- O *ranking* tem p elementos, $p \geq k$, ordenados, sem empates.
- Todos os elementos da lista também pertencem ao *ranking*.
- O último elemento do *ranking* sempre pertence à lista.
- As identidades dos elementos do *ranking* não importam — i.e., eles são indistinguíveis, a não ser por pertencerem ou não à lista (e pela ordem que ocupam no *ranking*, claro).

1.2 Criando *rankings*

1.2.1 Quantidade de *rankings*

Dados $k > 0$ e $p \geq k$ fixos, quantos *rankings* existem?

Para montar um *ranking*:

1. Sabemos que a última posição é ocupada por alguém da lista.
2. Só resta escolher as posições dos $k-1$ elementos restantes da lista dentre as $p-1$ posições restantes no *ranking*, o que dá $\binom{p-1}{k-1}$ escolhas.

Assim, a quantidade total de *rankings* para k e p dados é

$$\binom{p-1}{k-1}$$

1.2.2 Representação

Considere naturais $k > 0$ e $p \geq k$.

Podemos representar um *ranking* através de um *string* contendo k caracteres "`x`" e $p - k$ caracteres "`-`".

"`x`" representa uma posição ocupada por um elemento da lista.

"`-`" representa uma posição ocupada por um elemento que não está na lista.

Por exemplo, para $k = 3, p = 5$, os $\binom{4}{2} = 6$ *rankings* possíveis são

- `xx--x`
- `x-x-x`
- `x--xx`
- `-xx-x`
- `-x-xx`
- `--xxx`

A tabela a seguir (na verdade, um pedaço do triângulo de Pascal) mostra as quantidades de *rankings* possíveis para alguns valores de k e p:

	k									
p	1	2	3	4	5	6	7	8	9	10
1	1									
2	1	1								
3	1	2	1							
4	1	3	3	1						
5	1	4	6	4	1					
6	1	5	10	10	5	1				
7	1	6	15	20	15	6	1			
8	1	7	21	35	35	21	7	1		
9	1	8	28	56	70	56	28	8	1	
10	1	9	36	84	126	126	84	36	9	1
11	1	10	45	120	210	252	210	120	45	10
12	1	11	55	165	330	462	462	330	165	55
13	1	12	66	220	495	792	924	792	495	220
14	1	13	78	286	715	1.287	1.716	1.716	1.287	715
15	1	14	91	364	1.001	2.002	3.003	3.432	3.003	2.002
16	1	15	105	455	1.365	3.003	5.005	6.435	6.435	5.005
17	1	16	120	560	1.820	4.368	8.008	11.440	12.870	11.440
18	1	17	136	680	2.380	6.188	12.376	19.448	24.310	24.310
19	1	18	153	816	3.060	8.568	18.564	31.824	43.758	48.620

20	1	19	171	969	3.876	11.628	27.132	50.388	75.582	92.378
21	1	20	190	1.140	4.845	15.504	38.760	77.520	125.970	167.960
22	1	21	210	1.330	5.985	20.349	54.264	116.280	203.490	293.930
23	1	22	231	1.540	7.315	26.334	74.613	170.544	319.770	497.420
24	1	23	253	1.771	8.855	33.649	100.947	245.157	490.314	817.190
25	1	24	276	2.024	10.626	42.504	134.596	346.104	735.471	1.307.504
26	1	25	300	2.300	12.650	53.130	177.100	480.700	1.081.575	2.042.975
27	1	26	325	2.600	14.950	65.780	230.230	657.800	1.562.275	3.124.550
28	1	27	351	2.925	17.550	80.730	296.010	888.030	2.220.075	4.686.825
29	1	28	378	3.276	20.475	98.280	376.740	1.184.040	3.108.105	6.906.900
30	1	29	406	3.654	23.751	118.755	475.020	1.560.780	4.292.145	10.015.005

1.2.3 Criar um *ranking* a partir de um vetor

Em vez de especificar as p posições do *ranking*, pode ser mais compacto especificar as k posições do *ranking* que são ocupadas por elementos da lista.

A função `rk()` faz isso, recebendo um vetor numérico com k elementos e retornando um *string*.

```
rk(c(1, 3, 5, 7))
```

```
[1] "x-x-x-x"
```

Observe que as posições não precisam ser passadas em ordem:

```
rk(c(3, 7, 5, 1))
```

```
[1] "x-x-x-x"
```

A função detecta vetores que não podem representar *rankings*:

```
rk(c(3, 7, 3, 1))
```

```
Error in rk(c(3, 7, 3, 1)):
Valores precisam ser inteiros positivos, sem repetições.
```

```
rk(c(5, 7, 3, 1.5))
```

```
Error in rk(c(5, 7, 3, 1.5)):
Valores precisam ser inteiros positivos, sem repetições.
```

```
rk(c(5, -7, 3, 1))
```

```
Error in rk(c(5, -7, 3, 1)):
Valores precisam ser inteiros positivos, sem repetições.
```

1.3 Outras funções

1.3.1 Criar *plot*

A função `criar_plot` recebe um *ranking* e gera um gráfico de pontos, com um ponto para cada elemento.

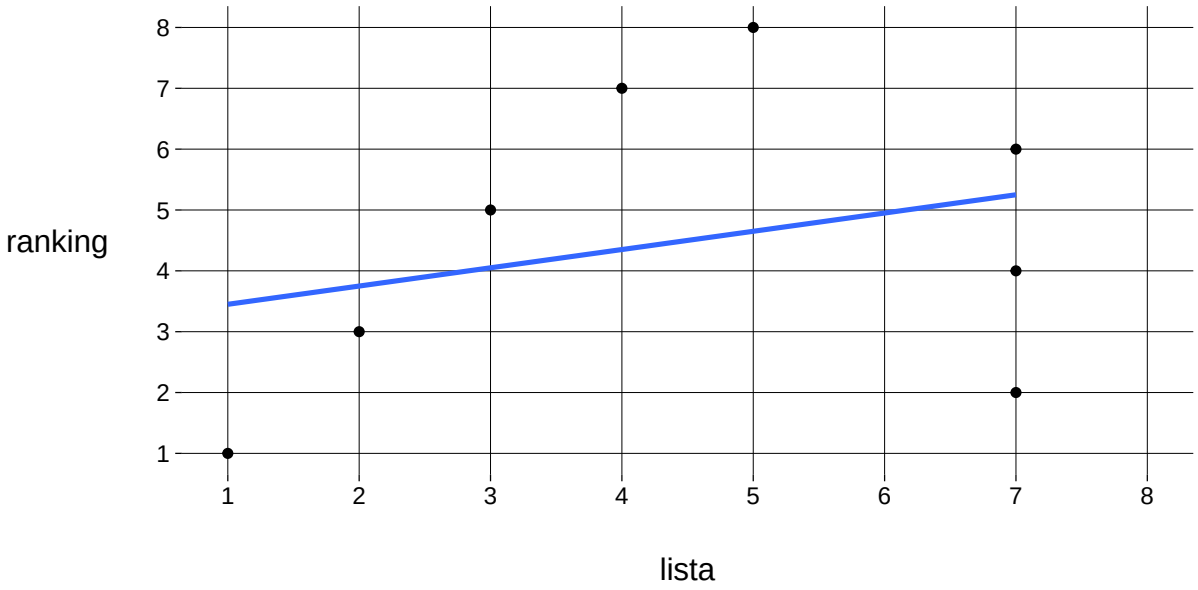
No eixo x, a posição do elemento na lista.

No eixo y, a posição do elemento no *ranking*.

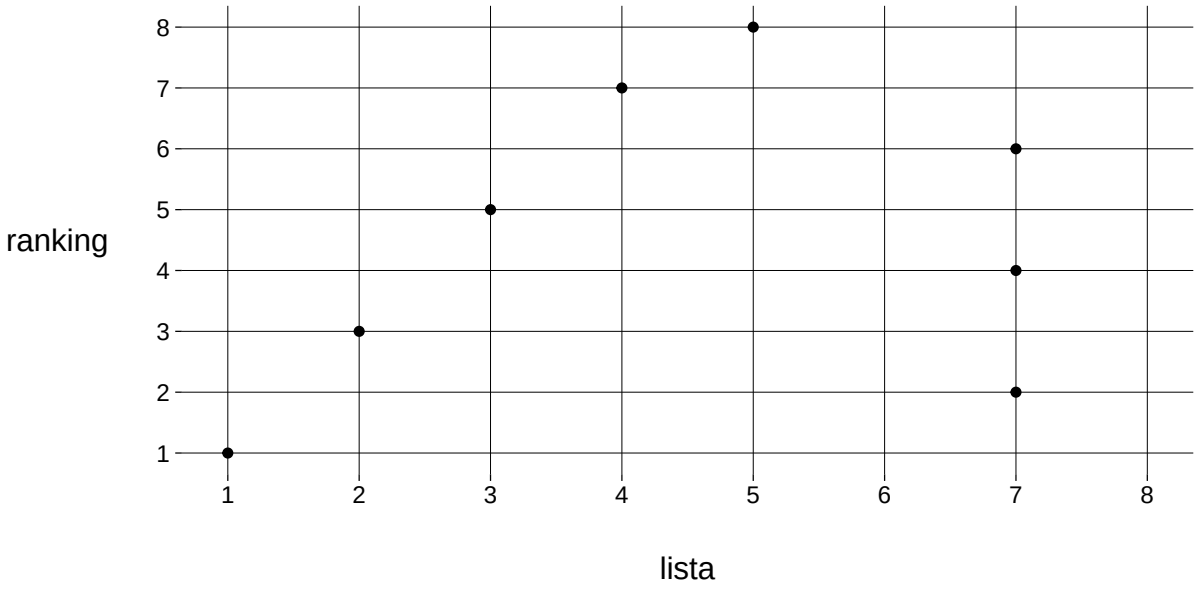
A função `criar_plot` pode receber um segundo argumento, opcional, especificando uma função para calcular o *score* deste *ranking* (i.e., alguma forma de correlação entre o *ranking* e a lista). O *score* vai ser mostrado no título do gráfico.

O terceiro argumento especifica se deve ser incluída uma reta de regressão linear via mínimos quadrados. O *default* é `TRUE`.

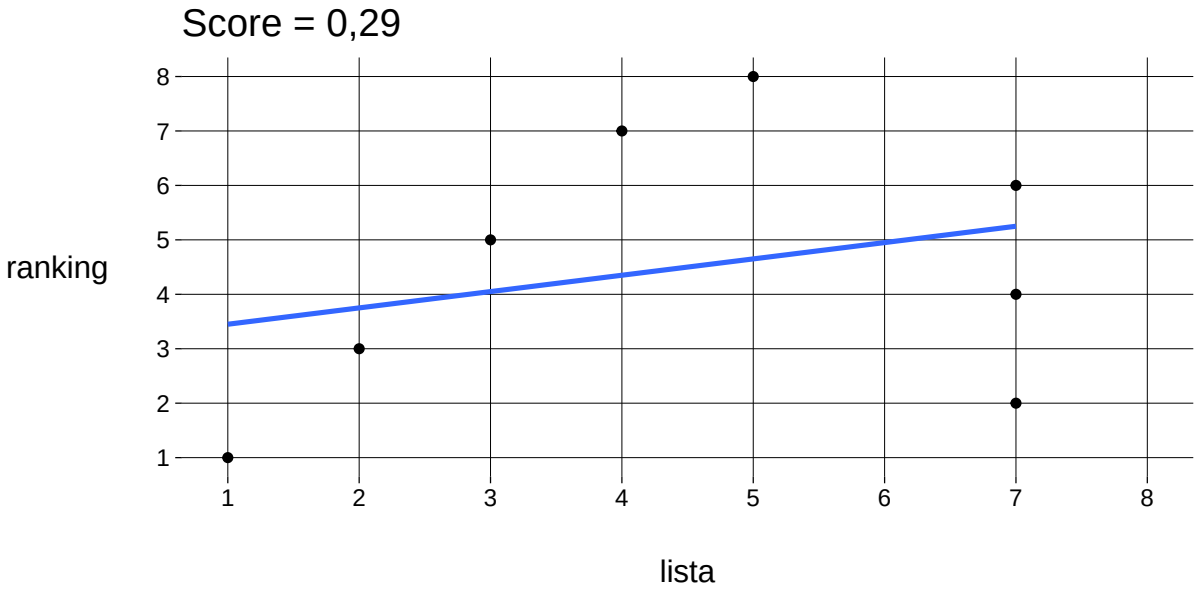
```
r <- 'x-x-x-xx'
criar_plot(r)
```




```
criar_plot(r, reta = FALSE)
```




```
criar_plot(r, \(df) { cor(df$pos_lista, df$pos_ranking) %>% round(2) })
```



1.3.2 Criar uma tibble com todos os *rankings*

Dados valores de p e k (nesta ordem), a função `criar_df_rankings()` retorna uma *tibble* com todos os $\binom{p-1}{k-1}$ *rankings* possíveis.

Se for passado apenas o valor de p, a função retorna uma *tibble* com todos os *rankings* possíveis de comprimento p (com k variando de 1 até p). Exercício: quantos são?

Cada *ranking* é representado por um *string*, como descrito na seção sobre a representação de *rankings*.

Todos os *rankings* com $p = 8$ e $k = 5$:

```
criar_df_rankings(8, 5)
```

```
# A tibble: 35 x 1
   ranking
   <chr>
 1 xxxx---x
 2 xxx-x--x
 3 xxx--x-x
 4 xxx---xx
 5 xx-xx--x
 6 xx-x-x-x
 7 xx-x--xx
 8 xx--xx-x
```

```
 9 xx--x-xx
10 xx---xxx
11 x-xxx--x
12 x-xx-x-x
13 x-xx--xx
14 x-x-xx-x
15 x-x-x-xx
16 x-x--xxx
17 x--xxx-x
18 x--xx-xx
19 x--x-xxx
20 x---xxxx
21 -xxxx--x
22 -xxx-x-x
23 -xxx--xx
24 -xx-xx-x
25 -xx-x-xx
26 -xx--xxx
27 -x-xxx-x
28 -x-xx-xx
29 -x-x-xxx
30 -x--xxxx
31 --xxxx-x
32 --xxx-xx
33 --xx-xxx
34 --x-xxxx
35 ---xxxxx
```

Todos os *rankings* com $p = 5$:

```
criar_df_rankings(5)
```

```
# A tibble: 16 x 1
   ranking
   <chr>
 1 ----x
 2 x---x
 3 -x--x
 4 --x-x
 5 ---xx
 6 xx--x
 7 x-x-x
```

```
 8 x--xx
 9 -xx-x
10 -x-xx
11 --xxx
12 xxx-x
13 xx-xx
14 x-xxx
15 -xxxx
16 xxxxx
```

2 O *ranking* concorda com a lista? Posições

2.1 Usando p como medida de concordância

Imagine que a lista de k elementos foi definida por uma autoridade, usando critérios que não conhecemos.

Em uma tentativa de descobrir esses critérios, construímos um modelo para avaliar todos os elementos da população (que inclui os k elementos da lista e outros).

Nosso modelo produz um *ranking* de todos os elementos. Para facilitar, vamos supor que não há empates no *ranking*.

Uma pergunta natural sobre a qualidade do *ranking* produzido é

> Quantas posições do *ranking* são necessárias para incluir todos os k elementos da lista?

A resposta é p, a posição, no *ranking*, do elemento da lista com pior classificação.

Aliás, é por isso que convencionamos, no capítulo anterior, que nossos *rankings* sempre terminam com um elemento da lista.

Um exemplo:

- A lista contém $k = 5$ elementos.
- O *ranking* r_1 é `xx-x-xx`, com $p = 7$.
- O *ranking* r_2 é `-xxxxx`, com $p = 6$.

Segundo a medida proposta aqui, r_2 é melhor que r_1.

Ou seja, quanto menor o valor de p, melhor o *ranking*.

Embora comparar *rankings* através de seus valores de p seja simples, podemos examinar medidas alternativas, que sejam mais finas que esta.

Por exemplo, é discutível se os dois rankings `xx---x` e `---xxx` devem ser considerados igualmente bons; no entanto, ambos têm $p = 6$.

2.2 Usando p e as posições dos elementos da lista

2.2.1 Contando posições –

Dado um *ranking* r com k e p, queremos definir uma função $s(r)$ com as seguintes características:

- Se r não contiver "`-`", então $s(r) = 1$. Neste caso, r é um *ranking* perfeito, que coincide com a lista (por exemplo, `xxxxx`). Em casos assim, $k = p$. Vamos definir s como sendo da forma

  $$s(r) = \frac{k}{p} + \cdots$$

  onde as reticências representam termos que ainda vamos definir. Se r for um *ranking* perfeito, a parcela k/p será 1, e vamos definir os termos restantes para que sejam iguais a zero.

- Os termos restantes devem ter valores maiores quanto melhor for o *ranking*. Quanto mais próximos do fim do *ranking* estiverem os caracteres "`-`", melhor ele será. Uma quantidade natural seria

  $$\frac{\text{soma\_}}{\sum_{i=1,p} i} \quad = \quad \frac{\text{soma\_}}{p(p+1)/2} \quad = \quad \frac{2\,\text{soma\_}}{p(p+1)}$$

  onde soma_ é a soma das posições ocupadas por "`_`" em r.

  Como queríamos, quando r for um *ranking* perfeito, soma_ $= 0$, e então $s(r) = 1$.

- Mas também queremos que somente *rankings* perfeitos tenham $s(r) = 1$. Para isso, considere que um *ranking* mais próximo do perfeito é da forma

  `x...x-x`

  Ou seja, $k = p - 1$ e soma_ $= p - 1$.

  Vamos multiplicar a segunda parcela por α de forma que $s(r) < 1$ para este *ranking* quase perfeito:

  $$s(r) = \frac{p-1}{p} + \frac{2(p-1)}{p(p+1)} \cdot \alpha$$

  Então

$$
\begin{aligned}
s(r) < 1 &\iff \frac{2(p-1)}{p(p+1)} \cdot \alpha < \frac{1}{p} \\
&\iff 2\alpha(p-1) < p+1 \\
&\iff \alpha < \frac{1}{2} \cdot \frac{p+1}{p-1} \\
&\iff \alpha = \frac{1}{m} \cdot \frac{p+1}{p-1} \qquad (m > 2)
\end{aligned}
$$

o que dá

$$
\begin{aligned}
s(r) &= \frac{k}{p} + \frac{2\,\text{soma\_}}{p(p+1)} \cdot \alpha \\
&= \frac{k}{p} + \frac{2\,\text{soma\_}}{p(p+1)} \cdot \frac{1}{m} \cdot \frac{p+1}{p-1} \qquad (m > 2) \\
&= \frac{k}{p} + \frac{2\,\text{soma\_}}{p(p-1)} \cdot \frac{1}{m} \qquad (m > 2) \\
&= \frac{k}{p} + \frac{\text{soma\_}}{p(p-1)} \cdot \frac{2}{m} \qquad (m > 2)
\end{aligned}
$$

Dependendo do valor de $m > 2$ escolhido, teremos medidas diferentes.

A função abaixo usa o *default* de $m = 10$, mas valores diferentes podem ser passados.

```
s <- function(ranking, m = 10) {

  # Vetor de caracteres
  ranking <- str_split(ranking, '')[[1]]

  p <- length(ranking)
  k <- sum(ranking == 'x')
  soma_ <- sum(which(ranking == '-'))

  (k / p) + ((2 * soma_) / (m * p * (p - 1)))

}

s <- Vectorize(s)
```

Para $p = 8$, alguns exemplos:

```r
s(
  c(
    'xxxxxxxx',
    'xxxxxx-x',
    '-xxxxxxx'
  )
)
```

```
 xxxxxxxx  xxxxxx-x  -xxxxxxx
1,0000000 0,9000000 0,8785714
```

Todos os *rankings* de comprimento 8, com suas pontuações:

```
# A tibble: 128 x 2
    ranking      s
    <chr>    <dbl>
  1 xxxxxxxx 1
  2 xxxxxx-x 0.9
  3 xxxxx-xx 0.896
  4 xxxx-xxx 0.893
  5 xxx-xxxx 0.889
  6 xx-xxxxx 0.886
  7 x-xxxxxx 0.882
  8 -xxxxxxx 0.879
  9 xxxxx--x 0.796
 10 xxxx-x-x 0.793
 11 xxxx--xx 0.789
 12 xxx-xx-x 0.789
 13 xxx-x-xx 0.786
 14 xx-xxx-x 0.786
 15 xxx--xxx 0.782
 16 xx-xx-xx 0.782
 17 x-xxxx-x 0.782
 18 xx-x-xxx 0.779
 19 x-xxx-xx 0.779
 20 -xxxxx-x 0.779
 21 xx--xxxx 0.775
 22 x-xx-xxx 0.775
 23 -xxxx-xx 0.775
 24 x-x-xxxx 0.771
 25 -xxx-xxx 0.771
```

```
26 x--xxxxx 0.768
27 -xx-xxxx 0.768
28 -x-xxxxx 0.764
29 --xxxxxx 0.761
30 xxxx---x 0.689
31 xxx-x--x 0.686
32 xxx--x-x 0.682
33 xx-xx--x 0.682
34 xxx---xx 0.679
35 xx-x-x-x 0.679
36 x-xxx--x 0.679
37 xx-x--xx 0.675
38 xx--xx-x 0.675
39 x-xx-x-x 0.675
40 -xxxx--x 0.675
41 xx--x-xx 0.671
42 x-xx--xx 0.671
43 x-x-xx-x 0.671
44 -xxx-x-x 0.671
45 xx---xxx 0.668
46 x-x-x-xx 0.668
47 x--xxx-x 0.668
48 -xxx--xx 0.668
49 -xx-xx-x 0.668
50 x-x--xxx 0.664
51 x--xx-xx 0.664
52 -xx-x-xx 0.664
53 -x-xxx-x 0.664
54 x--x-xxx 0.661
55 -xx--xxx 0.661
56 -x-xx-xx 0.661
57 --xxxx-x 0.661
58 x---xxxx 0.657
59 -x-x-xxx 0.657
60 --xxx-xx 0.657
61 -x--xxxx 0.654
62 --xx-xxx 0.654
63 --x-xxxx 0.65
64 ---xxxxx 0.646
65 xxx----x 0.579
66 xx-x---x 0.575
67 xx--x--x 0.571
68 x-xx---x 0.571
```

```
 69 xx---x-x 0.568
 70 x-x-x--x 0.568
 71 -xxx---x 0.568
 72 xx----xx 0.564
 73 x-x--x-x 0.564
 74 x--xx--x 0.564
 75 -xx-x--x 0.564
 76 x-x---xx 0.561
 77 x--x-x-x 0.561
 78 -xx--x-x 0.561
 79 -x-xx--x 0.561
 80 x--x--xx 0.557
 81 x---xx-x 0.557
 82 -xx---xx 0.557
 83 -x-x-x-x 0.557
 84 --xxx--x 0.557
 85 x---x-xx 0.554
 86 -x-x--xx 0.554
 87 -x--xx-x 0.554
 88 --xx-x-x 0.554
 89 x----xxx 0.55
 90 -x--x-xx 0.55
 91 --xx--xx 0.55
 92 --x-xx-x 0.55
 93 -x---xxx 0.546
 94 --x-x-xx 0.546
 95 ---xxx-x 0.546
 96 --x--xxx 0.543
 97 ---xx-xx 0.543
 98 ---x-xxx 0.539
 99 ----xxxx 0.536
100 xx-----x 0.464
101 x-x----x 0.461
102 x--x---x 0.457
103 -xx----x 0.457
104 x---x--x 0.454
105 -x-x---x 0.454
106 x----x-x 0.45
107 -x--x--x 0.45
108 --xx---x 0.45
109 x-----xx 0.446
110 -x---x-x 0.446
111 --x-x--x 0.446
```

```
112 -x----xx 0.443
113 --x--x-x 0.443
114 ---xx--x 0.443
115 --x---xx 0.439
116 ---x-x-x 0.439
117 ---x--xx 0.436
118 ----xx-x 0.436
119 ----x-xx 0.432
120 -----xxx 0.429
121 x------x 0.346
122 -x-----x 0.343
123 --x----x 0.339
124 ---x---x 0.336
125 ----x--x 0.332
126 -----x-x 0.329
127 ------xx 0.325
128 -------x 0.225
```

Perceba que pode haver empates: `xxxx--xx` e `xxx-xx-x` têm o mesmo valor de s. É razoável achar que estes dois *rankings* têm a mesma qualidade.

2.2.2 Comparando *rankings* com valores diferentes de p

Como a lista é dada e fixa, só faz sentido, na prática, comparar *rankings* com o mesmo valor de k.

Vamos examinar, para uma lista com $k = 15$, os *rankings* possíveis com p variando de 15 a 20.

São 15.504 *rankings*. Eis os 100 melhores:

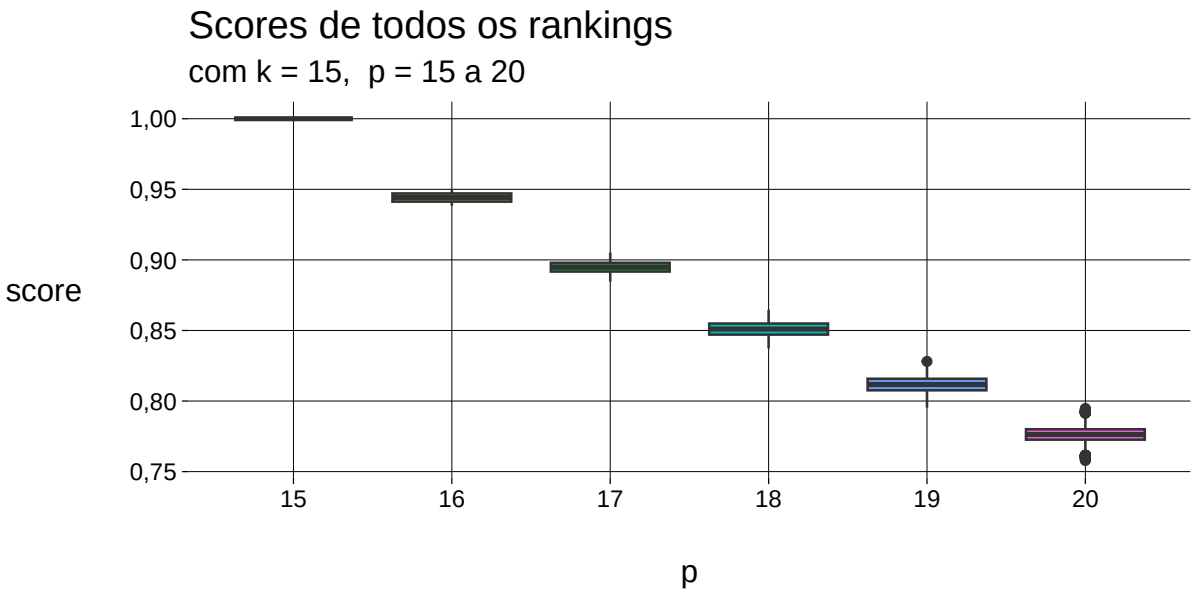
```
# A tibble: 100 x 3
    ranking               s     p
    <chr>             <dbl> <int>
  1 xxxxxxxxxxxxxxx   1        15
  2 xxxxxxxxxxxxxx-x  0.95     16
  3 xxxxxxxxxxxxx-xx  0.949    16
  4 xxxxxxxxxxxx-xxx  0.948    16
  5 xxxxxxxxxxx-xxxx  0.948    16
  6 xxxxxxxxxx-xxxxx  0.947    16
  7 xxxxxxxxx-xxxxxx  0.946    16
  8 xxxxxxxx-xxxxxxx  0.945    16
```

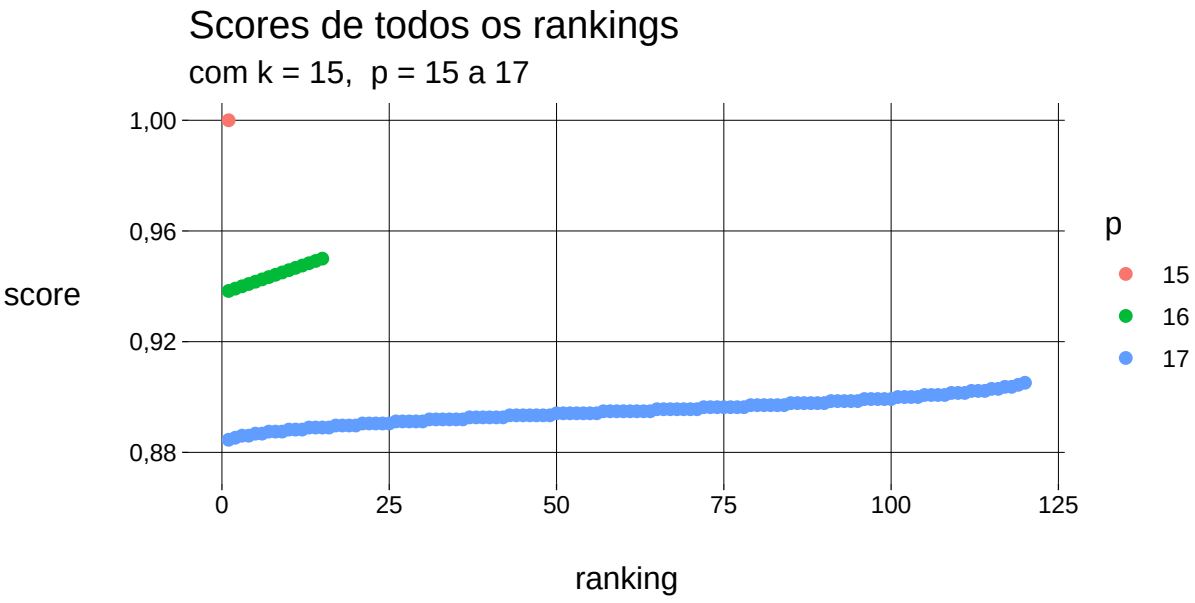
```
 9 xxxxxxx-xxxxxxxx  0.944    16
10 xxxxxx-xxxxxxxxx  0.943    16
11 xxxxx-xxxxxxxxxx  0.942    16
12 xxxx-xxxxxxxxxxx  0.942    16
13 xxx-xxxxxxxxxxxx  0.941    16
14 xx-xxxxxxxxxxxxx  0.94     16
15 x-xxxxxxxxxxxxxx  0.939    16
16 -xxxxxxxxxxxxxxx  0.938    16
17 xxxxxxxxxxxxxx--x 0.905    17
18 xxxxxxxxxxxxx-x-x 0.904    17
19 xxxxxxxxxxxxx--xx 0.904    17
20 xxxxxxxxxxxx-xx-x 0.904    17
21 xxxxxxxxxxxx-x-xx 0.903    17
22 xxxxxxxxxxx-xxx-x 0.903    17
23 xxxxxxxxxxx--xxx 0.902    17
24 xxxxxxxxxxx-xx-xx 0.902    17
25 xxxxxxxxxx-xxxx-x 0.902    17
26 xxxxxxxxxx-x-xxx 0.901    17
27 xxxxxxxxxx-xxx-xx 0.901    17
28 xxxxxxxxx-xxxxx-x 0.901    17
29 xxxxxxxxxx--xxxx 0.901    17
30 xxxxxxxxxx-xx-xxx 0.901    17
31 xxxxxxxxx-xxxx-xx 0.901    17
32 xxxxxxxx-xxxxxx-x 0.901    17
33 xxxxxxxxxx-x-xxxx 0.9      17
34 xxxxxxxxx-xxx-xxx 0.9      17
35 xxxxxxxx-xxxxx-xx 0.9      17
36 xxxxxxx-xxxxxxx-x 0.9      17
37 xxxxxxxxxx--xxxxx 0.899    17
38 xxxxxxxxx-xx-xxxx 0.899    17
39 xxxxxxxx-xxxx-xxx 0.899    17
40 xxxxxxx-xxxxxx-xx 0.899    17
41 xxxxxx-xxxxxxxx-x 0.899    17
42 xxxxxxxxx-x-xxxxx 0.899    17
43 xxxxxxxx-xxx-xxxx 0.899    17
44 xxxxxxx-xxxxx-xxx 0.899    17
45 xxxxxx-xxxxxxx-xx 0.899    17
46 xxxxx-xxxxxxxxx-x 0.899    17
47 xxxxxxxxx--xxxxxx 0.898    17
48 xxxxxxxx-xx-xxxxx 0.898    17
49 xxxxxxx-xxxx-xxxx 0.898    17
50 xxxxxx-xxxxxx-xxx 0.898    17
51 xxxxx-xxxxxxxx-xx 0.898    17
```

```
52 xxxx-xxxxxxxxxx-x 0.898    17
53 xxxxxxxx-x-xxxxxx 0.897    17
54 xxxxxxx-xxx-xxxxx 0.897    17
55 xxxxxx-xxxxx-xxxx 0.897    17
56 xxxxx-xxxxxxx-xxx 0.897    17
57 xxxx-xxxxxxxxx-xx 0.897    17
58 xxx-xxxxxxxxxxx-x 0.897    17
59 xxxxxxxx--xxxxxxx 0.896    17
60 xxxxxxx-xx-xxxxxx 0.896    17
61 xxxxxx-xxxx-xxxxx 0.896    17
62 xxxxx-xxxxxx-xxxx 0.896    17
63 xxxx-xxxxxxxx-xxx 0.896    17
64 xxx-xxxxxxxxxx-xx 0.896    17
65 xx-xxxxxxxxxxxx-x 0.896    17
66 xxxxxxx-x-xxxxxxx 0.896    17
67 xxxxxx-xxx-xxxxxx 0.896    17
68 xxxxx-xxxxx-xxxxx 0.896    17
69 xxxx-xxxxxxx-xxxx 0.896    17
70 xxx-xxxxxxxxx-xxx 0.896    17
71 xx-xxxxxxxxxx-xx 0.896    17
72 x-xxxxxxxxxxxx-x 0.896    17
73 xxxxxxx--xxxxxxxx 0.895    17
74 xxxxxx-xx-xxxxxxx 0.895    17
75 xxxxx-xxxx-xxxxxx 0.895    17
76 xxxx-xxxxxx-xxxxx 0.895    17
77 xxx-xxxxxxxx-xxxx 0.895    17
78 xx-xxxxxxxxx-xxx 0.895    17
79 x-xxxxxxxxxxx-xx 0.895    17
80 -xxxxxxxxxxxxx-x 0.895    17
81 xxxxxx-x-xxxxxxx 0.894    17
82 xxxxx-xxx-xxxxxxx 0.894    17
83 xxxx-xxxxx-xxxxxx 0.894    17
84 xxx-xxxxxxx-xxxxx 0.894    17
85 xx-xxxxxxxxx-xxxx 0.894    17
86 x-xxxxxxxxxx-xxx 0.894    17
87 -xxxxxxxxxxxx-xx 0.894    17
88 xxxxxx--xxxxxxxx 0.893    17
89 xxxxx-xx-xxxxxxx 0.893    17
90 xxxx-xxxx-xxxxxxx 0.893    17
91 xxx-xxxxxx-xxxxxx 0.893    17
92 xx-xxxxxxx-xxxxx 0.893    17
93 x-xxxxxxxxx-xxxx 0.893    17
94 -xxxxxxxxxxx-xxx 0.893    17
```

```
 95 xxxxx-x-xxxxxxxxx 0.893    17
 96 xxxx-xxx-xxxxxxxx 0.893    17
 97 xxx-xxxxx-xxxxxxx 0.893    17
 98 xx-xxxxxxx-xxxxxx 0.893    17
 99 x-xxxxxxxxx-xxxxx 0.893    17
100 -xxxxxxxxxxx-xxxx 0.893    17
```

Os gráficos abaixo mostram os *scores* atribuídos para todos os *rankings* com $k = 15$ e p variando de 15 a 20, separados por valores de p:




Scores de todos os rankings
com k = 15, p = 15 a 17
score
1,00
0,96
0,92
0,88
0
25
50
75
100
125
ranking
p
15
16
17


Scores de todos os rankings
com k = 15, p = 18 a 20
score
0,87
0,84
0,81
0,78
0,75
0
3000
6000
9000
12000
ranking
p
18
19
20