Projeto 1 - FLS 6497

Pedro Schmalz 10389052

2022-10-22

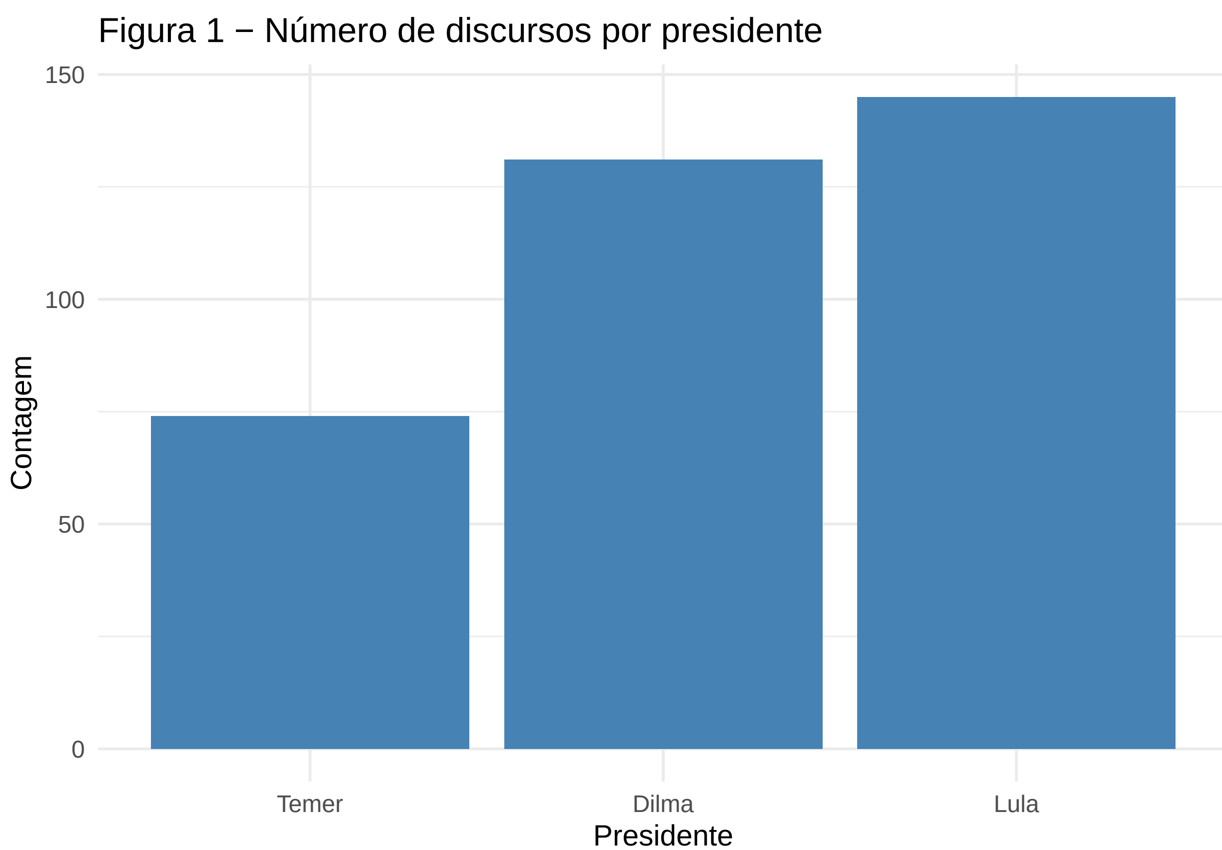
Prevendo a autoria de discursos presidenciais.

Neste trabalho, utilizaremos aprendizado de máquina (*machine learning*) para prever a autoria de discursos presidenciais.

Dados

Temos dois bancos distintos, um de treinamento e um de validação. Nestes bancos, há discursos de três presidentes: Lula, Dilma e Temer.

Nosso banco de treino se encontra dividido entre as classes da seguinte maneira:



Podemos ver que Temer é o presidente com menor número de discursos, mas a desproporcionalidade não parece ser grande o suficiente para gerar maiores problemas.

Table 2: Tabela 2 - Perfomance de cada pipeline em uma iteração

Pipeline	Accuracy	Bal. Acc.	Brier Score	Class. Error
baseline	0.5809524	0.6472416	0.8380952	0.4190476
baseline+stop	0.5809524	0.6462704	0.8380929	0.4190476
bigrams	0.4761905	0.5580808	1.0475860	0.5238095
bigrams+stopwords	0.4761905	0.5580808	1.0475860	0.5238095
trigrams	0.5047619	0.5827506	0.9858143	0.4952381
trigrams+stopwords	0.5047619	0.5827506	0.9858143	0.4952381

Testando diferentes pré-processamentos

Como forma de avaliar o impacto do pré-processamento na performance dos modelos, irei testar alguns pré-processamentos com um modelo (Naive-Bayes). As principais alterações no pré-processamento serão no número de ngrams (1, 2 e 3) e se há a opção do stopwords = "pt". Com isso, serão comparadas 3*2 = 6 pipelines diferentes de começo. A tabela 1 abaixo resume as diferentes pipelines:

Tabela 1 - Diferentes Pipelines de pré-processamento

Pipeline	ngram	stopwords
1 (Baseline)	1	não
2	1	sim
3	2	não
4	2	sim
5	3	não
6	3	sim

Benchmark (Pré-processamentos)

Nesta seção, faremos o benchmark dos pré-processamentos e compararemos os resultados do modelo utilizando o Naive Bayes como learner para todos os pré-processamentos. Devido à exigências do classificador, removi a coluna de data e transformei a variável de presidente em numérica, por ordem de mandato (Lula, 1; Dilma 2 e Temer, 3).

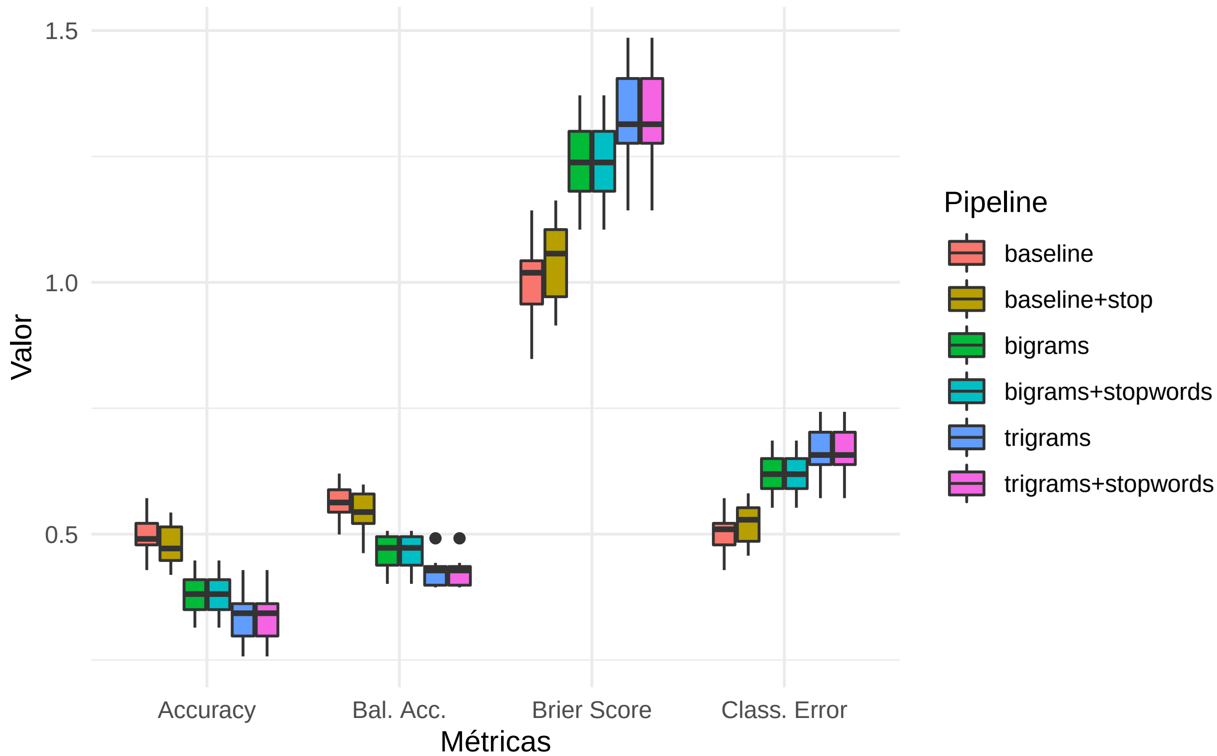
A tabela 2 mostra os resultados de uma iteração:

O pipeline que nos serve como baseline parece ter obtido o melhor resultado. De forma a confirmar isto, repetimos a operação 10x e computamos os resultados na figura 1:

Table 3: Tabela 3 - Perfomance de cada modelo em uma iteração

Modelo	Accuracy	Bal. Acc.	Brier Score	Class. Error
Naive Bayes (Baseline)	0.4952381	0.5575684	1.0095238	0.5047619
Tree	0.8285714	0.8140097	0.2267972	0.1714286
KNN	0.4380952	0.5060386	0.7647989	0.5619048
Forest	0.9904762	0.9907407	0.2449454	0.0095238

Figura 2 − Performance de cada pipeline



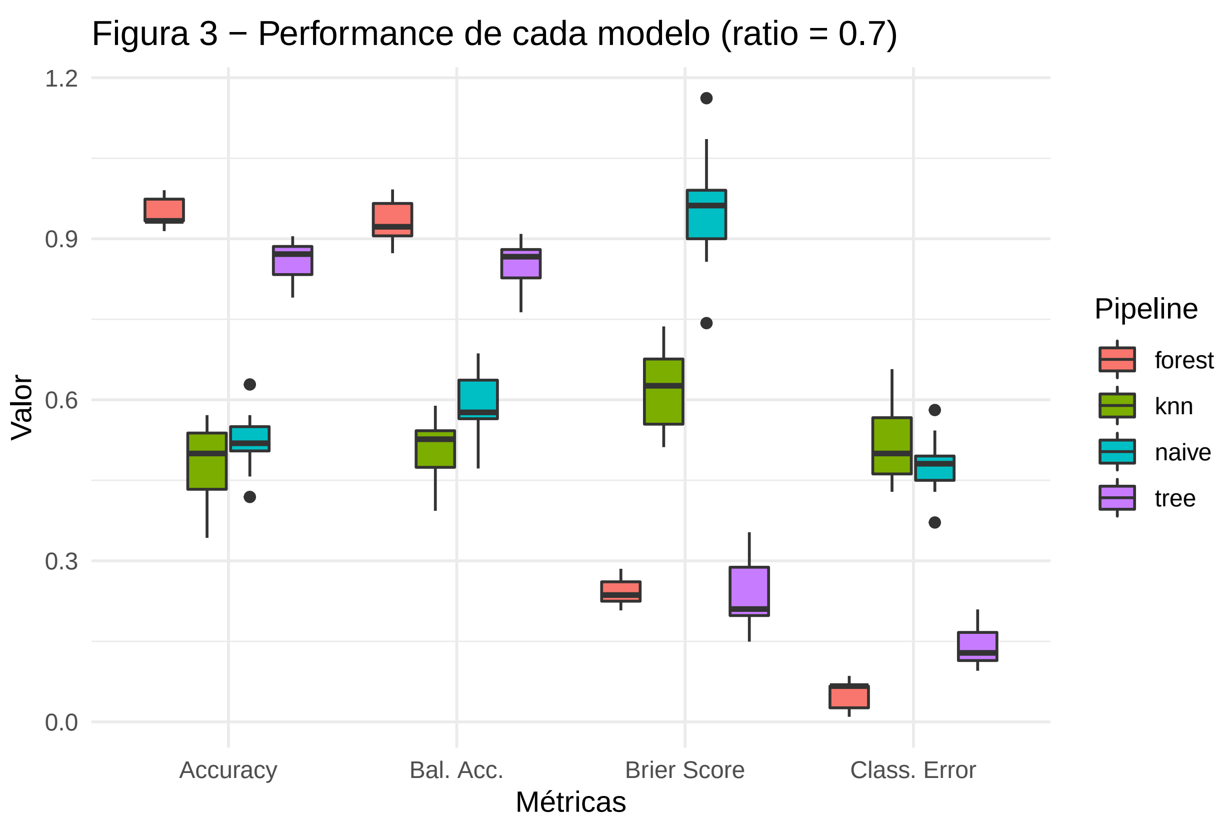
wer the Brier score is for a set of predictions, the better the predictions are calibrated.

o baseline com ngram = 1 e sem a correção de stop-words obteve melhores resultados. Portanto, ele será o utilizados para a comparação dos modelos. Na segunda parte, utilizaremos quatro modelos com o primeiro pipeline: o Naive Bayes, Tree, K-nearest Neighbors, e Random Forest. A tabela 3 mostra os resultados dos modelos em uma iteração.

Novamente, confirmamos o resultado em uma simulação de 10 iterações, representado na figura 2

Table 4: Tabela 4 - Perfomance de cada modelo em uma iteração

Modelo	Accuracy	Bal. Acc.	Brier Score	Class. Error
naive	0.5047619	0.5477477	0.9904762	0.4952381
naive (bag)	0.4666667	0.5137387	0.9962928	0.5333333
naive (sub)	0.4571429	0.5040541	1.0460724	0.5428571
tree	0.8095238	0.7991634	0.2908215	0.1904762
tree (bag)	0.8380952	0.8257079	0.1785543	0.1619048
tree (sub)	0.8857143	0.8867761	0.2067254	0.1142857
knn	0.5333333	0.5135779	0.6091935	0.4666667
knn (bag)	0.5619048	0.5564350	0.5667111	0.4380952
knn (sub)	0.5047619	0.4736165	0.5833834	0.4952381
ranger	0.9714286	0.9700772	0.2605040	0.0285714
ranger (bag)	0.9333333	0.9282497	0.2872199	0.0666667
ranger (sub)	0.9619048	0.9552767	0.2954229	0.0380952
rf	0.9714286	0.9671815	0.2540152	0.0285714
rf (bag)	0.8095238	0.7719434	0.3209010	0.1904762
rf (sub)	0.8952381	0.8848777	0.2879333	0.1047619



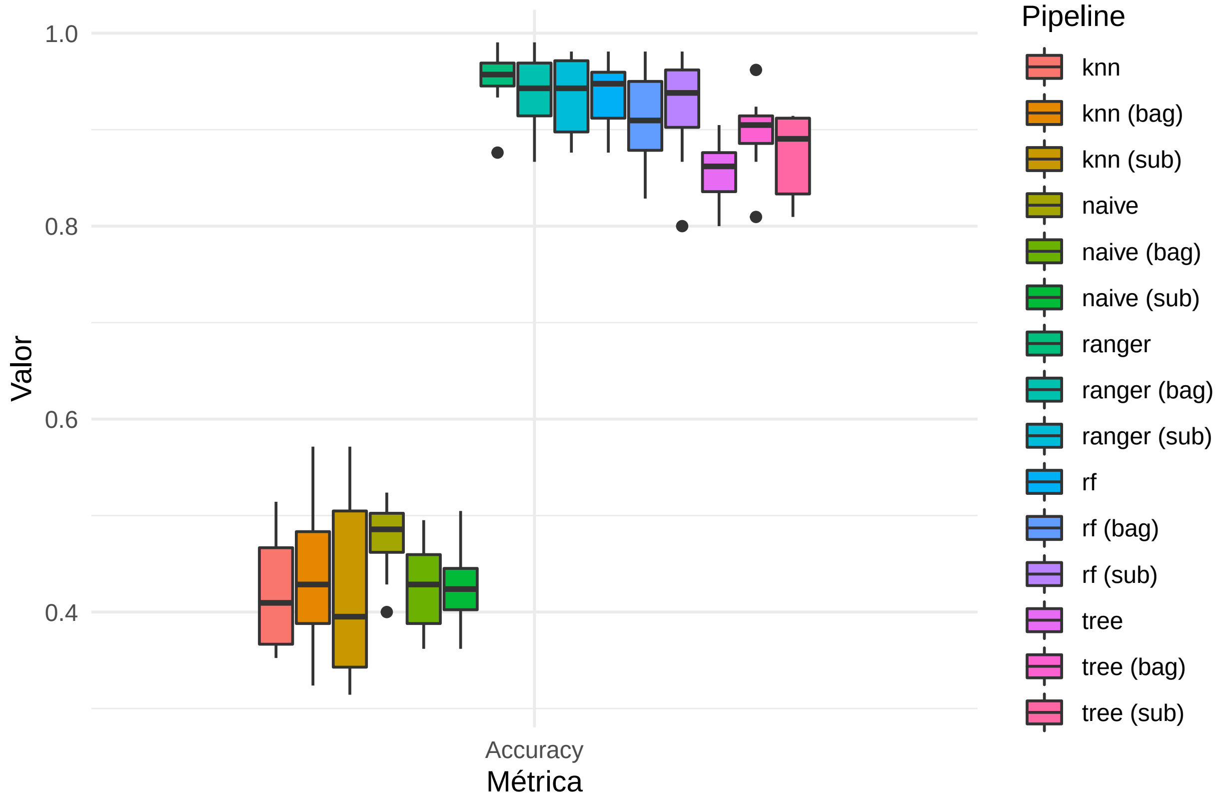
Note: the lower the Brier score is for a set of predictions, the better the predictions are calibrated.

Bagging

Pegaremos os modelos (forest e tree), e testaremos eles contra alguns baggings ensembles (com e sem subsample) :

Os resultados de uma iteração aparecem na tabela 4:

Figura 4 − Performance de cada modelo (Accuracy)



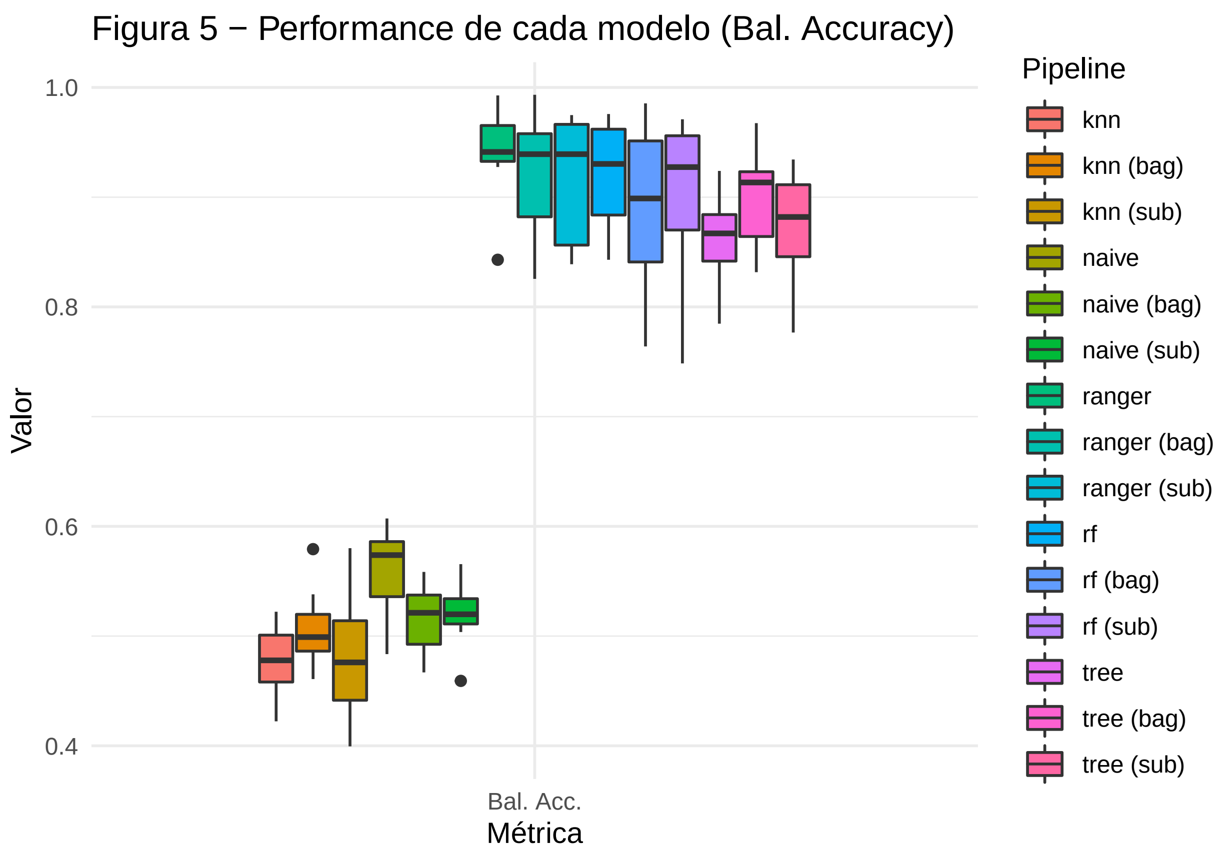
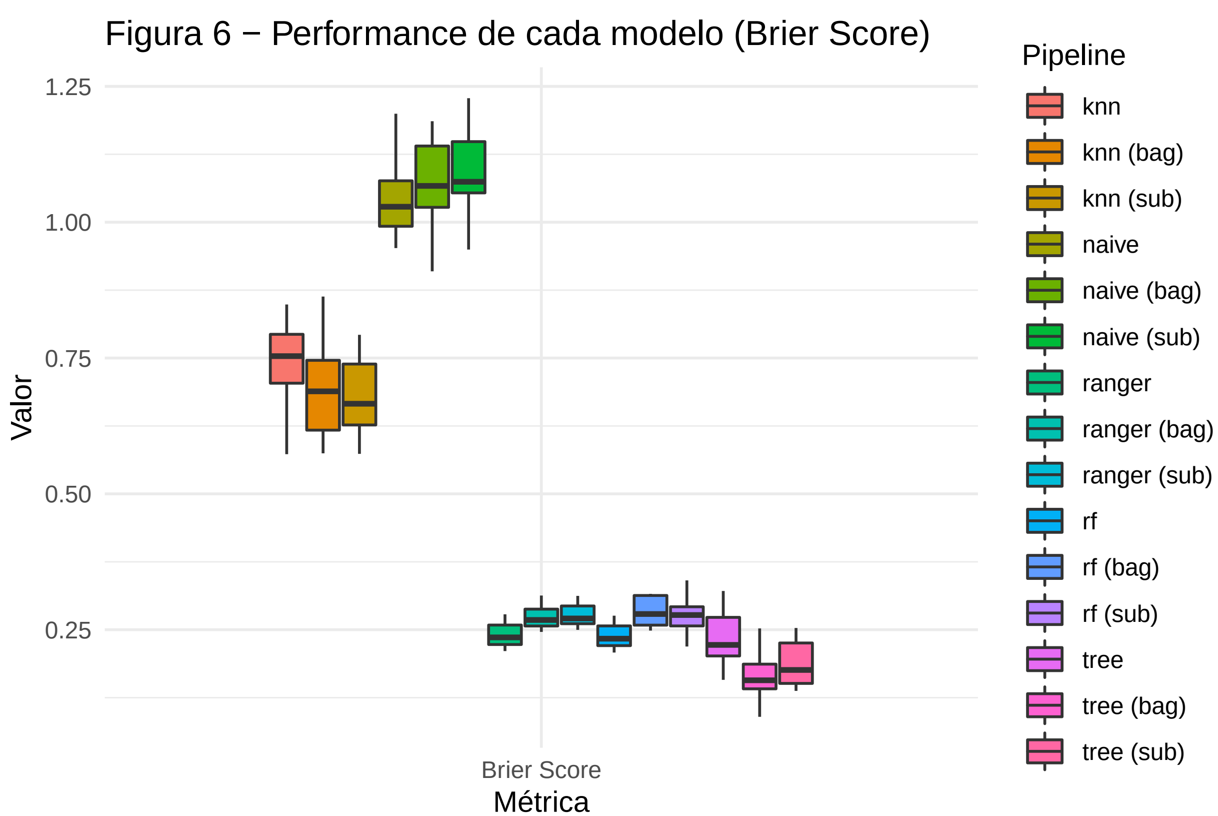


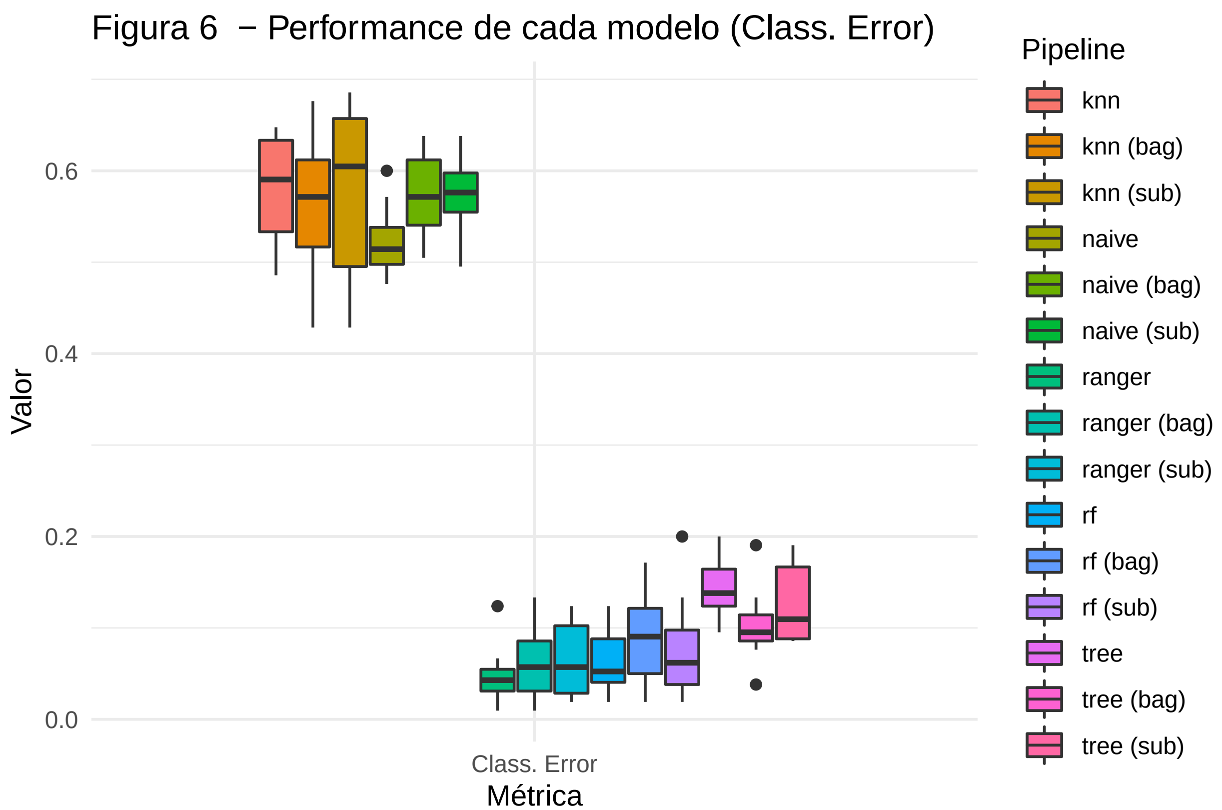
Figura 5 − Performance de cada modelo (Bal. Accuracy)
Valor
1.0
0.8
0.6
0.4
Bal. Acc.
Métrica
Pipeline
knn
knn (bag)
knn (sub)
naive
naive (bag)
naive (sub)
ranger
ranger (bag)
ranger (sub)
rf
rf (bag)
rf (sub)
tree
tree (bag)
tree (sub)



e: the lower the Brier score is for a set of predictions, the better the predictions are calibrated.

Table 5: Tabela 5 - Perfomance do stacking em uma iteração

Modelo	Accuracy	Bal. Acc.	Brier Score	Class. Error
ensemble	0.9619048	0.9602049	0.0760896	0.0380952



e: the lower the Brier score is for a set of predictions, the better the predictions are calibrated.

Ensemble em cima de ensemble

Iremos fazer um stack dos três melhores modelos nas simulações(ranger, ranger(bag), rf), com o multinomial log-linear model como agregador:

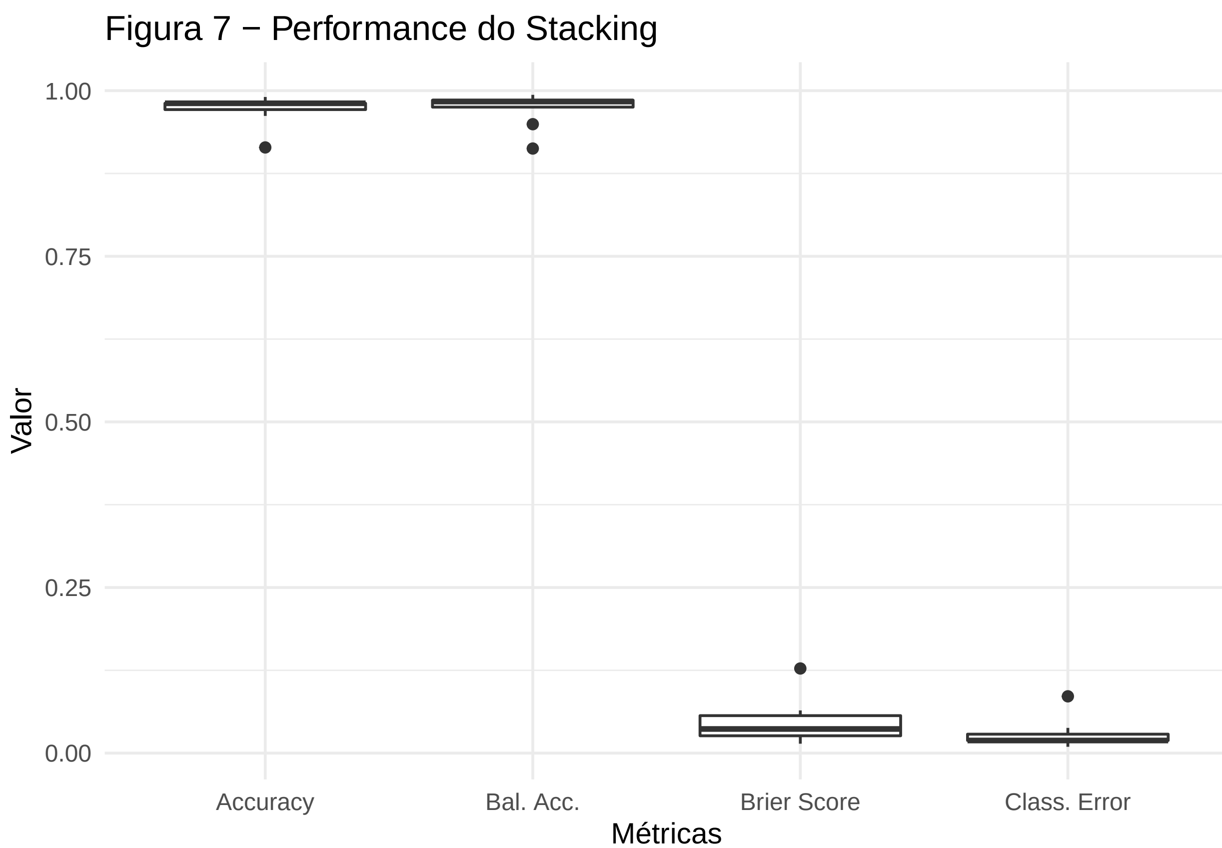
```
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 17.818126
## iter  20 value 14.995110
## iter  30 value 13.396115
## iter  40 value 11.972165
## iter  50 value 11.066879
## iter  60 value 10.410415
## iter  70 value 10.030882
## iter  80 value 9.592428
## iter  90 value 8.690847
## iter 100 value 8.516982
## final  value 8.516982
## stopped after 100 iterations
```

Novamente, faremos uma simulação para ver se há muita variação nos resultados do modelo

```
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 33.001019
## iter  20 value 28.032723
## iter  30 value 26.479532
## iter  40 value 25.822560
## iter  50 value 25.761262
## iter  60 value 25.749271
## iter  70 value 25.739539
## iter  80 value 25.737602
## iter  90 value 25.737320
## iter 100 value 25.737218
## final  value 25.737218
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 35.366981
## iter  20 value 25.905084
## iter  30 value 24.724870
## iter  40 value 24.085678
## iter  50 value 23.774627
## iter  60 value 23.697745
## iter  70 value 23.597060
## iter  80 value 23.559050
## iter  90 value 23.543097
## iter 100 value 23.540217
## final  value 23.540217
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 20.704679
## iter  20 value 17.382591
## iter  30 value 15.511461
## iter  40 value 15.051549
## iter  50 value 14.952735
## iter  60 value 14.783614
## iter  70 value 14.740001
## iter  80 value 14.660245
## iter  90 value 14.455351
## iter 100 value 14.440308
## final  value 14.440308
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 20.242507
## iter  20 value 17.093270
## iter  30 value 16.223802
## iter  40 value 15.930455
## iter  50 value 15.807156
## iter  60 value 15.592597
## iter  70 value 15.429712
## iter  80 value 15.038612
## iter  90 value 14.963086
## iter 100 value 14.845981
```

```
## final  value 14.845981
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 28.594819
## iter  20 value 24.948642
## iter  30 value 23.856440
## iter  40 value 23.713043
## iter  50 value 23.678274
## iter  60 value 23.646255
## iter  70 value 23.636646
## iter  80 value 23.627138
## iter  90 value 23.625476
## iter 100 value 23.625319
## final  value 23.625319
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 36.362800
## iter  20 value 29.605657
## iter  30 value 29.054008
## iter  40 value 28.947950
## iter  50 value 28.932835
## iter  60 value 28.926495
## iter  70 value 28.924804
## iter  80 value 28.924600
## final  value 28.924585
## converged
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 35.660392
## iter  20 value 27.256552
## iter  30 value 26.086807
## iter  40 value 25.650756
## iter  50 value 25.589284
## iter  60 value 25.501849
## iter  70 value 25.493761
## iter  80 value 25.493081
## iter  90 value 25.492831
## iter 100 value 25.492765
## final  value 25.492765
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 16.628817
## iter  20 value 13.072945
## iter  30 value 11.939847
## iter  40 value 11.750272
## iter  50 value 11.712878
## iter  60 value 11.653496
## iter  70 value 11.630904
## iter  80 value 11.616479
## iter  90 value 11.608738
## iter 100 value 11.606418
```

```
## final  value 11.606418
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 26.989633
## iter  20 value 21.837023
## iter  30 value 20.301467
## iter  40 value 19.353089
## iter  50 value 19.047309
## iter  60 value 18.633683
## iter  70 value 18.453685
## iter  80 value 18.238211
## iter  90 value 18.016960
## iter 100 value 17.898743
## final  value 17.898743
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 16.679913
## iter  20 value 10.460145
## iter  30 value 7.190383
## iter  40 value 5.926092
## iter  50 value 5.175813
## iter  60 value 4.658294
## iter  70 value 4.276987
## iter  80 value 4.026751
## iter  90 value 3.940545
## iter 100 value 3.864942
## final  value 3.864942
## stopped after 100 iterations
```



Boosting

Por fim, compararemos os resultados do stacking feito com os modelos: o gradient boosting e o extreme gradient boosting:

```
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 28.161084
## iter  20 value 21.441317
## iter  30 value 19.241650
## iter  40 value 18.674571
## iter  50 value 18.430525
## iter  60 value 18.180654
## iter  70 value 18.048382
## iter  80 value 17.890892
## iter  90 value 17.877966
## iter 100 value 17.868848
## final  value 17.868848
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
```

Parece que o Extreme boosting obteve resultados levemente superiores ao stacking. Agora, testaremos isso novamente em uma simulação:

```
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 20.701398
```

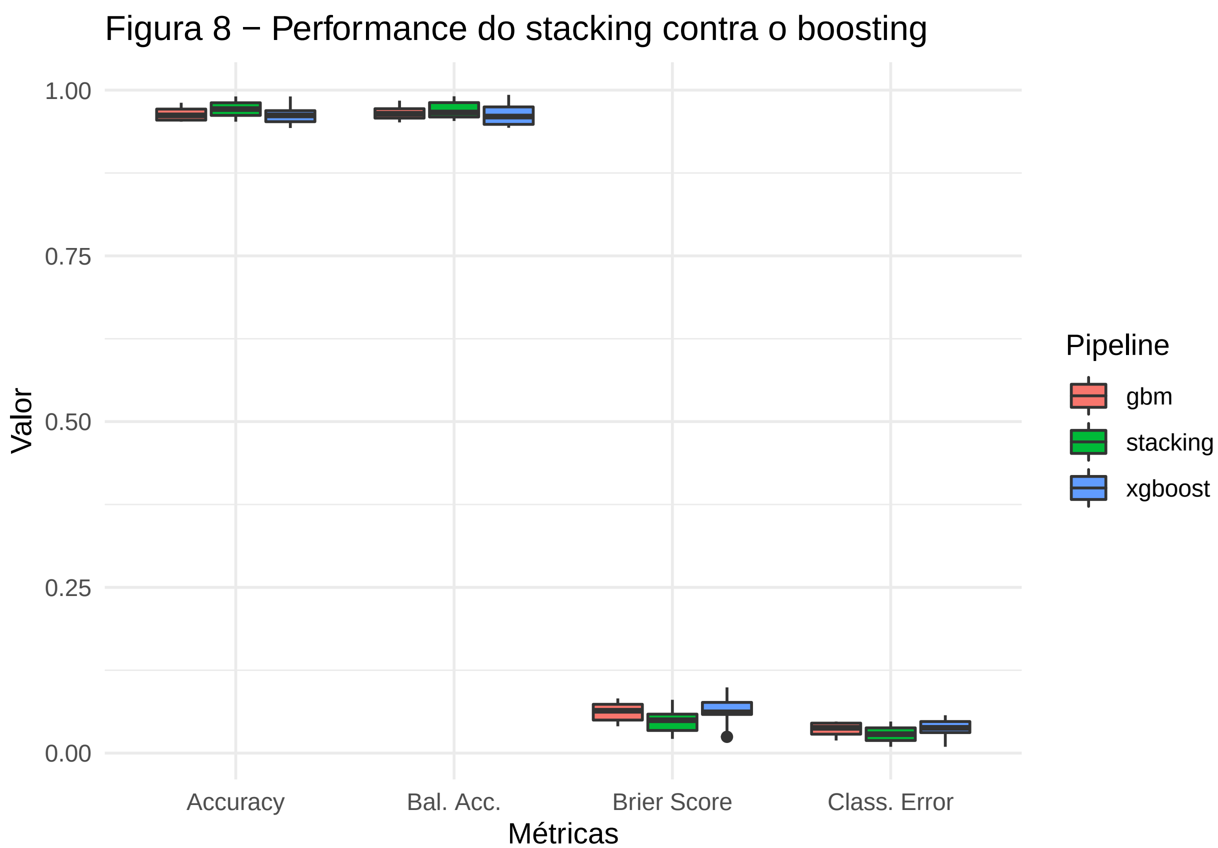
Table 6: Tabela 6 - Perfomance do stacking contra o boosting

Modelo	Accuracy	Bal. Acc.	Brier Score	Class. Error
stacking	0.9619048	0.9603175	0.0557679	0.0380952
xgboost	0.9809524	0.9823232	0.0443650	0.0190476
gbm	0.9809524	0.9779942	0.0433728	0.0190476

```
## iter  20 value 14.428519
## iter  30 value 12.528425
## iter  40 value 11.031454
## iter  50 value 10.813241
## iter  60 value 10.585197
## iter  70 value 10.480281
## iter  80 value 10.437285
## iter  90 value 10.433529
## iter 100 value 10.419742
## final  value 10.419742
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 23.120175
## iter  20 value 19.921929
## iter  30 value 19.362205
## iter  40 value 19.260595
## iter  50 value 19.224773
## iter  60 value 19.167664
## iter  70 value 19.138566
## iter  80 value 19.118385
## iter  90 value 19.111457
## iter 100 value 19.109605
## final  value 19.109605
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 33.684564
## iter  20 value 27.310486
## iter  30 value 26.231421
## iter  40 value 25.982604
## iter  50 value 25.846945
## iter  60 value 25.813020
## iter  70 value 25.796510
## iter  80 value 25.792004
## iter  90 value 25.790061
## iter 100 value 25.789834
## final  value 25.789834
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 26.947399
## iter  20 value 21.159926
```

```
## iter  30 value 19.111152
## iter  40 value 18.485315
## iter  50 value 18.196728
## iter  60 value 17.995136
## iter  70 value 17.786546
## iter  80 value 17.566454
## iter  90 value 17.502093
## iter 100 value 17.483750
## final  value 17.483750
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 31.399519
## iter  20 value 25.789648
## iter  30 value 25.597852
## iter  40 value 25.526121
## iter  50 value 25.516022
## iter  60 value 25.512435
## iter  70 value 25.508599
## iter  80 value 25.508055
## iter  90 value 25.507982
## iter 100 value 25.507964
## final  value 25.507964
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 24.188819
## iter  20 value 18.466474
## iter  30 value 17.445907
## iter  40 value 16.897475
## iter  50 value 16.456155
## iter  60 value 16.318814
## iter  70 value 16.189356
## iter  80 value 16.049025
## iter  90 value 15.991708
## iter 100 value 15.970385
## final  value 15.970385
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 21.651488
## iter  20 value 19.004205
## iter  30 value 18.000766
## iter  40 value 17.503851
## iter  50 value 17.356674
## iter  60 value 17.313118
## iter  70 value 17.266287
## iter  80 value 17.259700
## iter  90 value 17.259402
## iter 100 value 17.259276
## final  value 17.259276
```

```
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 42.452823
## iter  20 value 37.287900
## iter  30 value 36.894722
## iter  40 value 36.866519
## iter  50 value 36.858903
## iter  60 value 36.856714
## iter  70 value 36.855923
## iter  80 value 36.855777
## final  value 36.855773
## converged
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 24.531336
## iter  20 value 21.853736
## iter  30 value 20.837642
## iter  40 value 20.589149
## iter  50 value 20.543463
## iter  60 value 20.533503
## iter  70 value 20.525131
## iter  80 value 20.522561
## iter  90 value 20.521544
## iter 100 value 20.521433
## final  value 20.521433
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
## # weights:  33 (20 variable)
## initial  value 269.160011
## iter  10 value 27.482467
## iter  20 value 24.277502
## iter  30 value 22.516338
## iter  40 value 21.696576
## iter  50 value 21.651000
## iter  60 value 21.616052
## iter  70 value 21.593932
## iter  80 value 21.578401
## iter  90 value 21.575528
## iter 100 value 21.574342
## final  value 21.574342
## stopped after 100 iterations
## Distribution not specified, assuming multinomial ...
```



O stacking parece ter sido o modelo com melhores resultados. Portanto, ele será o utilizado para o treinamento e validação.

Como forma de melhorar a validação, iremos alterar o ratio do holdout para verificar a perfomance do modelo de stacking em diferentes tamanhos de bancos de treino.

```
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 26.154800
## iter  20 value 20.985019
## iter  30 value 18.927919
## iter  40 value 18.162139
## iter  50 value 17.947182
## iter  60 value 17.816496
## iter  70 value 17.777358
## iter  80 value 17.720373
## iter  90 value 17.682573
## iter 100 value 17.671097
## final  value 17.671097
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 37.170409
## iter  20 value 33.661623
## iter  30 value 32.357763
## iter  40 value 32.039581
## iter  50 value 31.937807
```

```
## iter  60 value 31.901150
## iter  70 value 31.870186
## iter  80 value 31.853078
## iter  90 value 31.846619
## iter 100 value 31.845378
## final  value 31.845378
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 25.406174
## iter  20 value 19.833686
## iter  30 value 19.231489
## iter  40 value 18.829829
## iter  50 value 18.689682
## iter  60 value 18.627746
## iter  70 value 18.604123
## iter  80 value 18.590392
## iter  90 value 18.578338
## iter 100 value 18.576189
## final  value 18.576189
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 23.829255
## iter  20 value 19.750257
## iter  30 value 18.777922
## iter  40 value 18.111543
## iter  50 value 17.533562
## iter  60 value 17.321654
## iter  70 value 17.064940
## iter  80 value 16.942618
## iter  90 value 16.872085
## iter 100 value 16.865462
## final  value 16.865462
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 31.162520
## iter  20 value 24.266837
## iter  30 value 21.756078
## iter  40 value 20.945209
## iter  50 value 20.838534
## iter  60 value 20.763899
## iter  70 value 20.718685
## iter  80 value 20.700266
## iter  90 value 20.682078
## iter 100 value 20.672497
## final  value 20.672497
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 27.004349
## iter  20 value 20.776578
## iter  30 value 18.359822
```

```
## iter  40 value 17.148030
## iter  50 value 16.054542
## iter  60 value 15.731817
## iter  70 value 15.574053
## iter  80 value 15.438864
## iter  90 value 15.368525
## iter 100 value 15.335156
## final  value 15.335156
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 17.764796
## iter  20 value 10.387205
## iter  30 value 9.024390
## iter  40 value 8.124066
## iter  50 value 8.005261
## iter  60 value 7.859092
## iter  70 value 7.567728
## iter  80 value 7.388049
## iter  90 value 7.312878
## iter 100 value 7.283476
## final  value 7.283476
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 27.897602
## iter  20 value 23.416500
## iter  30 value 22.479275
## iter  40 value 21.982498
## iter  50 value 21.844193
## iter  60 value 21.736880
## iter  70 value 21.687263
## iter  80 value 21.648453
## iter  90 value 21.516984
## iter 100 value 21.467513
## final  value 21.467513
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 27.316058
## iter  20 value 21.562409
## iter  30 value 20.235959
## iter  40 value 19.900371
## iter  50 value 19.830002
## iter  60 value 19.808059
## iter  70 value 19.795064
## iter  80 value 19.789181
## iter  90 value 19.786013
## iter 100 value 19.785975
## final  value 19.785975
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 307.611441
## iter  10 value 25.016606
```

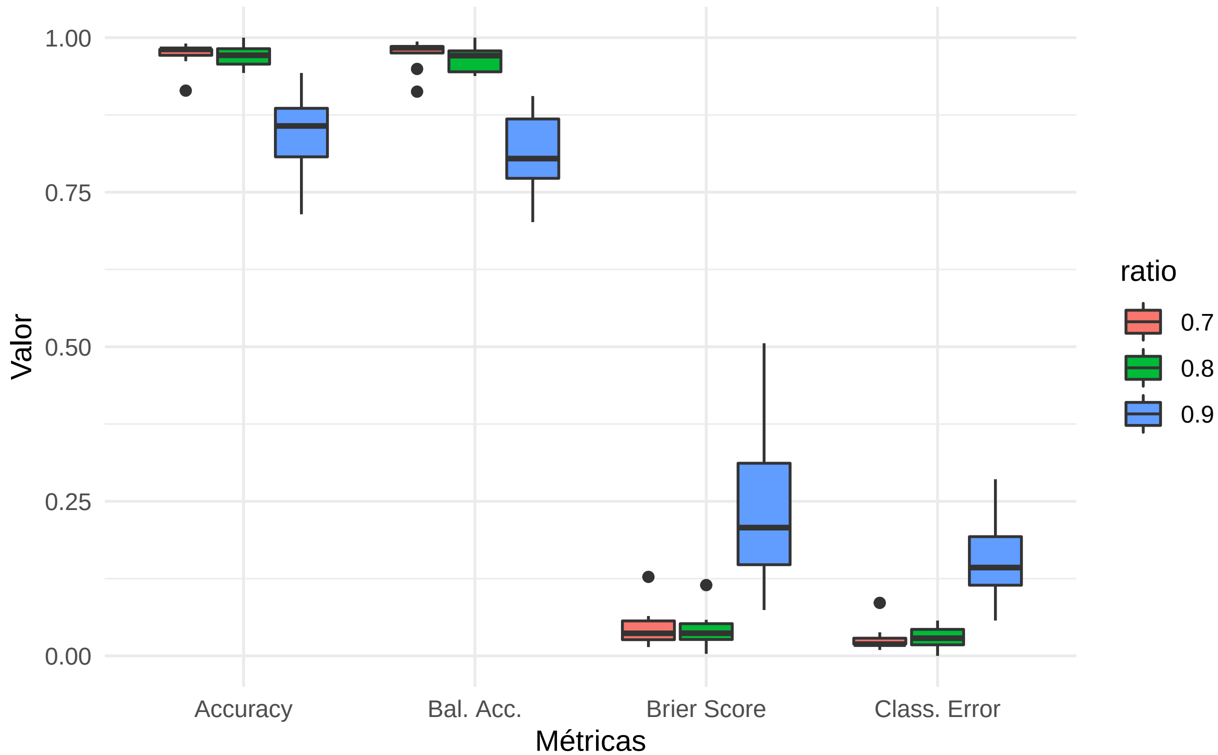
```
## iter  20 value 19.974441
## iter  30 value 18.797063
## iter  40 value 17.539220
## iter  50 value 17.298117
## iter  60 value 17.248084
## iter  70 value 17.231271
## iter  80 value 17.218075
## iter  90 value 17.212375
## iter 100 value 17.210383
## final  value 17.210383
## stopped after 100 iterations
```

```
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 21.415150
## iter  20 value 15.175232
## iter  30 value 14.312252
## iter  40 value 13.800488
## iter  50 value 13.614728
## iter  60 value 13.452921
## iter  70 value 13.162906
## iter  80 value 13.131807
## iter  90 value 13.114315
## iter 100 value 13.080712
## final  value 13.080712
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 22.446956
## iter  20 value 18.750637
## iter  30 value 17.987291
## iter  40 value 17.752203
## iter  50 value 17.744740
## iter  60 value 17.739598
## iter  70 value 17.732353
## iter  80 value 17.728890
## iter  90 value 17.728705
## iter 100 value 17.728405
## final  value 17.728405
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 21.581747
## iter  20 value 14.772003
## iter  30 value 13.953903
## iter  40 value 13.545150
## iter  50 value 13.216794
## iter  60 value 13.115258
## iter  70 value 12.995303
## iter  80 value 12.933702
## iter  90 value 12.900207
## iter 100 value 12.884597
## final  value 12.884597
## stopped after 100 iterations
```

```
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 24.580399
## iter  20 value 17.397546
## iter  30 value 16.484434
## iter  40 value 16.344682
## iter  50 value 16.306928
## iter  60 value 16.295786
## iter  70 value 16.288323
## iter  80 value 16.285204
## iter  90 value 16.284469
## iter 100 value 16.283925
## final  value 16.283925
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 28.158995
## iter  20 value 23.971589
## iter  30 value 23.275381
## iter  40 value 22.989811
## iter  50 value 22.890168
## iter  60 value 22.870709
## iter  70 value 22.820520
## iter  80 value 22.787609
## iter  90 value 22.775859
## iter 100 value 22.773804
## final  value 22.773804
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 18.357739
## iter  20 value 14.470120
## iter  30 value 13.440519
## iter  40 value 12.851241
## iter  50 value 12.682639
## iter  60 value 12.434946
## iter  70 value 12.228299
## iter  80 value 12.135557
## iter  90 value 12.068624
## iter 100 value 12.035821
## final  value 12.035821
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 27.473471
## iter  20 value 22.593992
## iter  30 value 21.800628
## iter  40 value 21.078204
## iter  50 value 21.000657
## iter  60 value 20.905008
## iter  70 value 20.866299
## iter  80 value 20.820940
## iter  90 value 20.805976
## iter 100 value 20.793275
```

```
## final  value 20.793275
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 24.608731
## iter  20 value 15.539913
## iter  30 value 14.570676
## iter  40 value 14.049302
## iter  50 value 13.872969
## iter  60 value 13.678946
## iter  70 value 13.455904
## iter  80 value 13.413755
## iter  90 value 13.369457
## iter 100 value 13.352959
## final  value 13.352959
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 25.043642
## iter  20 value 22.033798
## iter  30 value 21.535899
## iter  40 value 21.338198
## iter  50 value 21.298510
## iter  60 value 21.274287
## iter  70 value 21.266866
## iter  80 value 21.262205
## iter  90 value 21.261228
## iter 100 value 21.260960
## final  value 21.260960
## stopped after 100 iterations
## # weights:  33 (20 variable)
## initial  value 346.062871
## iter  10 value 24.101122
## iter  20 value 19.896126
## iter  30 value 18.501371
## iter  40 value 17.353242
## iter  50 value 17.123990
## iter  60 value 17.015327
## iter  70 value 16.961960
## iter  80 value 16.884818
## iter  90 value 16.843668
## iter 100 value 16.839226
## final  value 16.839226
## stopped after 100 iterations
```

Figura 9 − Performance do Forest por ratio



Note: the lower the Brier score is for a set of predictions, the better the predictions are calibrated.

Validação

O modelo stack com o ratio de 0.7 parece ser nosso melhor modelo. Avaliaremos seus resultados no banco de validação:

```
## # weights:  33 (20 variable)
## initial  value 384.514301
## iter  10 value 26.626072
## iter  20 value 17.457095
## iter  30 value 15.712443
## iter  40 value 14.449174
## iter  50 value 14.068127
## iter  60 value 13.888873
## iter  70 value 13.492118
## iter  80 value 13.211798
## iter  90 value 13.180795
## iter 100 value 13.061553
## final  value 13.061553
## stopped after 100 iterations
```

```
## # A tibble: 25 x 4
##    discurso                                                       id pred  presi~1
##    <chr>                                                       <dbl> <fct> <chr>
##  1 "\nExcelentíssimo senhor Shinzo Abe, primeiro-ministro d~   137 2     Dilma
##  2 "\nFoto: Roberto Stuckert Filho/PR \n \nSenhor Laurent F~    90 2     Dilma
##  3 "\nExcelentíssimo senhor Paul Biya, presidente do Camero~   229 1     Lula
```

Table 7: Tabela 6 - Predições do modelo stack (random forest)

id	pred	presidente
137	2	Dilma
90	2	Dilma
229	1	Lula
7	3	Temer
91	2	Dilma
364	1	Lula
153	2	Dilma
256	1	Lula
254	1	Lula
374	1	Lula
348	1	Lula
328	1	Lula
78	1	Lula
211	2	Dilma
118	2	Dilma
355	1	Lula
359	1	Lula
195	1	Lula
299	1	Lula
179	1	Lula
14	3	Temer
197	2	Dilma
306	1	Lula
26	1	Lula
244	1	Lula

```
##  4 "\nEu quero cumprimentar, em primeiro lugar, a senhora M~     7 3     Temer
##  5 "\nQuero dirigir um cumprimento especial à Zoleka Mandel~    91 2     Dilma
##  6 "\nExcelentíssimo Senhor Álvaro Uribe, Presidente da Col~   364 1     Lula
##  7 "\nSão passados mais de cinco anos do início da crise fi~   153 2     Dilma
##  8 "\nÉ um grande prazer iniciar a primeira visita oficial ~   256 1     Lula
##  9 "\n\n\nQuero agradecer ao presidente Ahmadinejad pela ho~   254 1     Lula
## 10 "\nMeu caro amigo Raúl Castro, Presidente da República d~   374 1     Lula
## # ... with 15 more rows, and abbreviated variable name 1: presidente
```