Projeto 1

Tamiris Burin

1. Pacotes necessários e a base de dados

Para este projeto, utilizamos as seguintes *libraries* e bancos de dados

```
library(tidyverse)
library(knitr)
library(kableExtra)
library(mlr3verse)
library(quanteda)
library(janitor)
library(kknn)
library(randomForest)
library(xgboost)
library(mlr3extralearners)
```

```
link <- "https://github.com/FLS-6497/datasets/blob/main/projeto1/discursos_pres_internacionais.csv?raw=
discursos <- readr::read_csv2(link) %>%
  mutate(id = row_number())

linkvalidacao <- "https://github.com/FLS-6497/datasets/blob/main/projeto1/discursos_pres_internacionais_
validacao <- readr::read_csv2(linkvalidacao) %>%
  mutate(id = row_number())
```

2. Modulando Processamentos

Construímos 3 *pipelines* de processamento:

A primeira utilizando as funções:

- *tolower*: que minimiza a caixa alta de todas palavras para a comparação.
- *remove_punct*: que remove a pontuação quando normalmente o contexto do corpus não é analisado.
- *remove_numbers*: que remove os números por nao serem relevantes tambem para a análise.
- *min_termfreq*: que estipula um valor mínimo de frequência do termo em todos os documentos.
- *stopwords_language = "pt"*: que remove as palavras pequenas e genericas chamadas de 'stopwords', como 'o', 'a', 'e' etc. por diferenciarem o conteúdo dos textos. Lembrando que usamos o banco de dados de stopwords com a função get_stopwords() para a língua apropriada.

A segunda utilizando todas da primeira e as seguintes:

- *stem*: que remove diferentes derivacoes dos termos, retornando os termos em seus radicais. Muito util para neutralizar plurais e verbos conjugados.

- *n=2*: que trata da identificação de *ngrams*. Os *ngrams* representam sequências frequentes de palavras do texto. Podemos especificar que procuramos por cunjuntos de 2 ou mais palavras e estes conjuntos, uma vez identificados, serão analisados como palavras/termos únicos.

E a terceira utilizando todas da anterior, porém substituindo a função de *n_gram* por:

- *scheme_df = 'inverse'*: que podendera o peso das ocorrências de uma palavra pela frequência em que ela aparece em um documento. Isso normaliza os elementos da matriz *bag of words*. Selecionando o tipo *inverse*, é calculada uma proporção da frequência inversa dos termos no documento.

```
processamento1 <-
  po("textvectorizer",
     tolower = TRUE,
     remove_punct = TRUE,
     remove_numbers = TRUE,
     min_termfreq = 30,
     stopwords_language = "pt") %>>%
  po("scale") %>>%
  po("mutate") %>>%
  po("learner", learner = lrn("classif.naive_bayes")) %>%
  as_learner()

processamento2 <-
  po("textvectorizer",
     tolower = TRUE,
     remove_punct = TRUE,
     remove_numbers = TRUE,
     min_termfreq = 30,
     stopwords_language = "pt",
     stem = TRUE,
     n=2) %>>%
  po("scale") %>>%
  po("mutate") %>>%
  po("learner", learner = lrn("classif.naive_bayes")) %>%
  as_learner()

processamento3 <-
  po("textvectorizer",
     tolower = TRUE,
     remove_punct = TRUE,
     remove_numbers = TRUE,
     min_termfreq = 30,
     stopwords_language = "pt",
     stem = TRUE,
     scheme_df = 'inverse') %>>%
  po("scale") %>>%
  po("mutate") %>>%
  po("learner", learner = lrn("classif.naive_bayes")) %>%
  as_learner()
```

3. Comparando Pré-Processamentos

Agora comparamos os três os pipelines de processamento aplicados à classificação do ***Naive Bayes*** junto da validação ***Accuracy***.

```
tsk <- as_task_classif(presidente ~ discurso, data = discursos)

  design.text <- benchmark_grid(
  tasks = tsk,
  learners = list(processamento1, processamento2, processamento3),
  resamplings = rsmp("holdout", ratio = 0.7))

resultados.text <- benchmark(design.text)
```

```
resultados.text$score(msr(c("classif.acc")))
```

Considerando que os pipelines de processamento n. 1 e n. 3 apresentaram os melhores resultados, rodamos 20 vezes cada com três métricas de validações diferentes: ***Accuracy***, ***Balanced Accuracy***, e o ***Classification Error***.

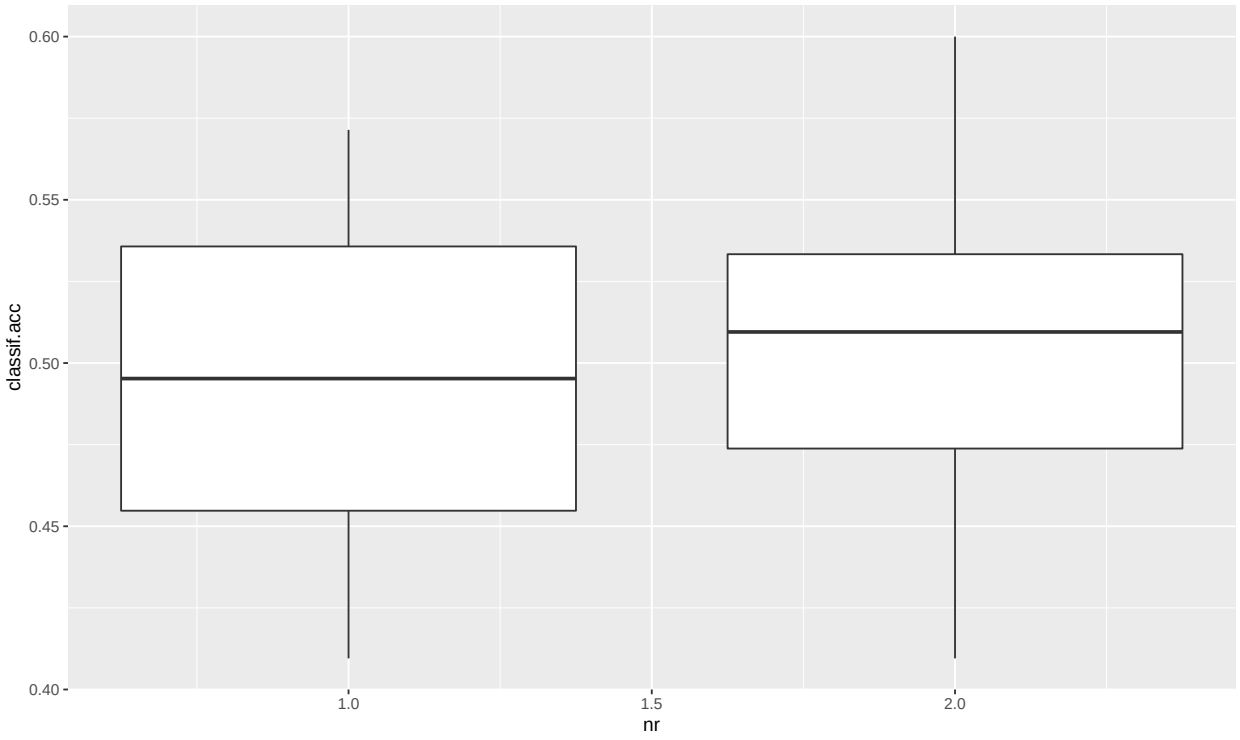
Foram gerados gráficos para as três métricas citadas.

Lembrando que a interpretação do Classification Error é que quanto menor é seu índice, mais acurado é o resultado.

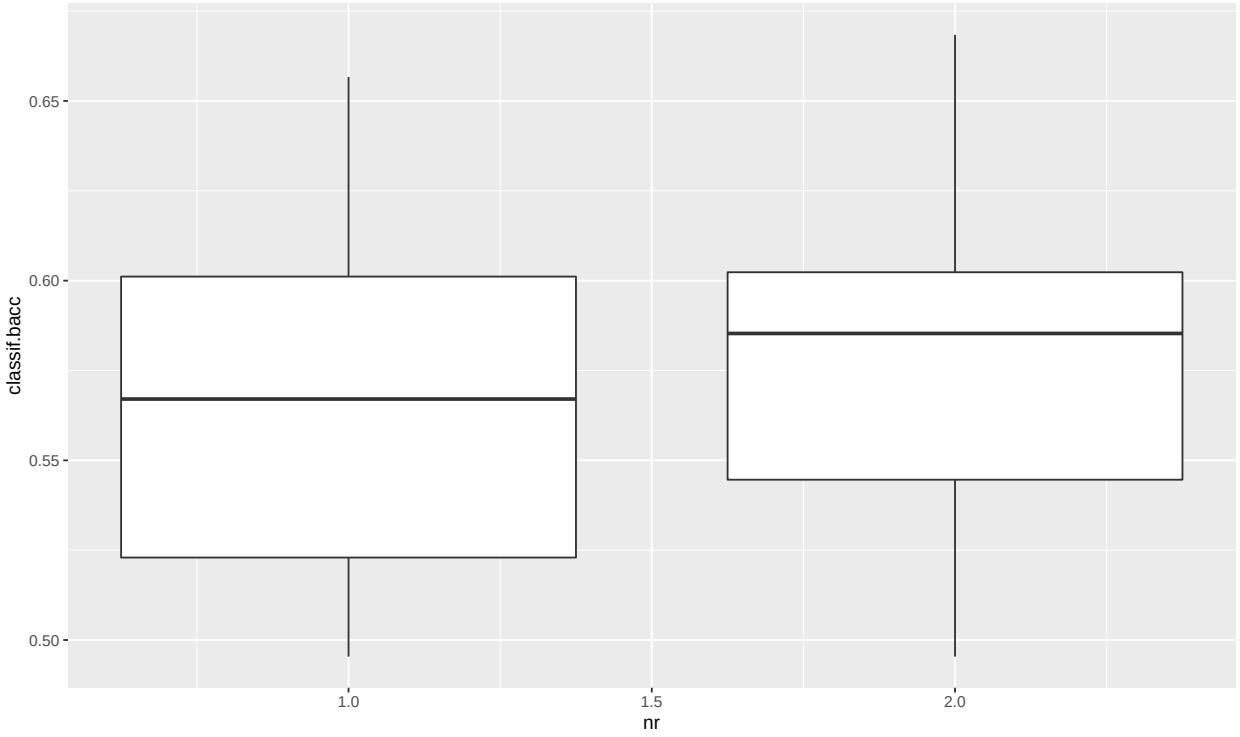
```
tsk <- as_task_classif(presidente ~ discurso, data = discursos)

validacaooprocessamentos <- function(){
  design <- benchmark_grid(
  tasks = tsk,
  learners = list(processamento1, processamento3),
  resamplings = rsmp("holdout", ratio = 0.7))

resultados_pipe <- benchmark(design)
resultados_pipe$score(msrs(c("classif.acc", "classif.bacc", "classif.ce")))
}
```

```
resultados_processamentos <- 1:20 %>%
  map_df(~ validacaooprocessamentos())
```

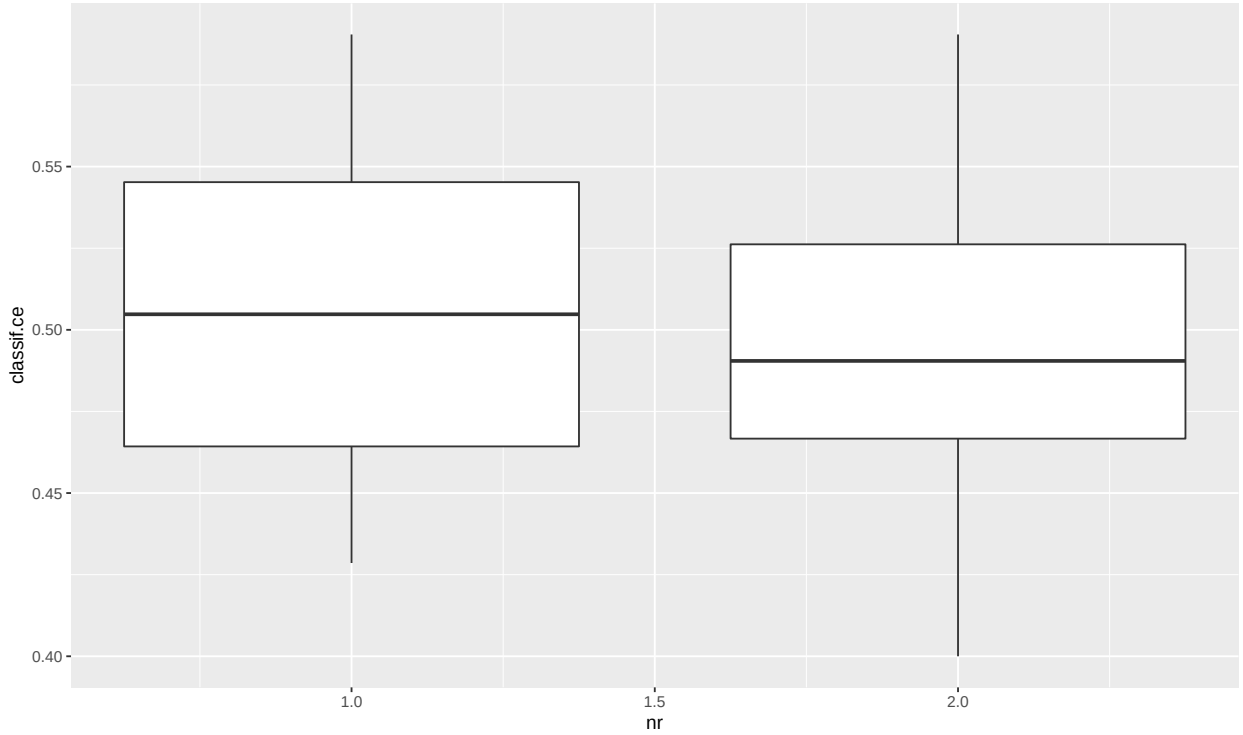
```
#Accuracy
resultados_processamentos %>%
  ggplot(aes(group = nr, y = classif.acc, x = nr)) +
  geom_boxplot()
```




```
#Balanced Accuracy
resultados_processamentos %>%
  ggplot(aes(group = nr, y = classif.bacc, x = nr)) +
  geom_boxplot()
```



```
#Classification Error
resultados_processamentos %>%
  ggplot(aes(group = nr, y = classif.ce, x = nr)) +
  geom_boxplot()
```



Verificamos que a *pipeline* de pré-processameno 3 é a melhor escolha, pois tem maior acurácia e menor erro, em média.

4. Pré-Processamento elegido

Enfim, aplicamos na base de discursos o *pipeline* de pré-processamento n. 3 elegido anteriormente:

```
processamento <- function(df, var) {
# Cria um corpus
cps <- corpus(df, text_field = var)

# Tokenizacao
tks <-
  tokens(cps, remove_punct = TRUE, remove_numbers = TRUE) %>%
  tokens_wordstem() %>%
  tokens_tolower() %>%
  tokens_remove(pattern = stopwords("pt"), min_nchar=4)

# Criacao de uma matriz bag-of-words
bow <- dfm(tks) %>%
  dfm_trim(min_termfreq = 30)%>%
  dfm_tfidf(scheme_df = "inverse")
```

```
# Transformar o resultado para tibble para o mlr3
dados <- as.matrix(bow) %>%
  as_tibble() %>%
  janitor::clean_names() %>%
  mutate_all(as.numeric)

# Definição do target
dados$y <- df$presidente

# Resultado da matrix
return(list(df = dados, bow = bow))

}
processamento(df=discursos, var="discurso")
```

5. Adequando as bases de Treino e Teste

Agora fazermos o *split sample* de validação, que neste caso é criar amostras de teste e de treino.

```
#Separando amostras de treino e teste, 70% e 30% respectivamente
discursostreino <- discursos %>%
  sample_frac(0.7)
discursosteste <- discursos %>%
  filter(!id %in% discursostreino$id)

#Processando treino
discursostreinobow <- processamento(df=discursostreino, var= "discurso")

#Adequando teste
discursostestebow <- discursosteste %>%
  corpus(text_field = "discurso") %>%
  tokens() %>%
  dfm() %>%
  dfm_match(featnames(discursostreinobow$bow)) %>%
  as.matrix() %>%
  as_tibble() %>%
  janitor::clean_names()%>%
  mutate_all(as.numeric)
discursostestebow$y <- as.factor(discursosteste$presidente)

#Treino
tsk <- as_task_classif(y ~.,data=discursostreinobow$df)
```

6. Modelos

Agora, com as amostras de treino e teste propriamente adequadas, aplicaremos três modelos de classificação para tentarmos predizer o nome do ou da presidente que proferiu cada discurso na amostra de teste:

1. ***Naive Bayes***

```
naive <-
  po("learner", learner = lrn("classif.naive_bayes"))  %>%
  as_learner()
```

2. ***k-Nearest Neighbors (KNN)***

```
kknn <-
  po("learner", learner = lrn("classif.kknn"))  %>%
  as_learner()
```

3. ***Random Forest***

```
forest <-
  po("learner", learner = lrn("classif.randomForest", ntree = 100))  %>%
  as_learner()
```

```
modelos <- benchmark_grid(
  tasks = tsk,
  learners = list(naive, kknn, forest),
  resamplings = rsmp("holdout", ratio = 0.7))

resultadosmodelos <- benchmark(modelos)
```

```
resultadosmodelos$score(msrs(c("classif.acc", "classif.bacc", "classif.ce")))
```

```
##                                  uhash nr              task
## 1: 3877092b-82a5-4489-896d-7da845239a9d  1 <TaskClassif[50]>
## 2: 0e30f3b8-bf62-432a-a3c3-e29537ea7c95  2 <TaskClassif[50]>
## 3: e31eff75-0c0d-4f73-920f-12e136b535e5  3 <TaskClassif[50]>
##                 task_id            learner          learner_id
## 1: discursostreinobow$df <GraphLearner[38]>  classif.naive_bayes
## 2: discursostreinobow$df <GraphLearner[38]>         classif.kknn
## 3: discursostreinobow$df <GraphLearner[38]> classif.randomForest
##                resampling resampling_id iteration              prediction
## 1: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
## 2: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
## 3: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
##    classif.acc classif.bacc classif.ce
## 1:   0.6986301    0.7371914 0.30136986
## 2:   0.5890411    0.6154321 0.41095890
## 3:   0.9589041    0.9654321 0.04109589
```

Podemos verificar que o modelo ***Random Forest*** apresentou os melhores resultados.

7. Aplicando Ensembles

Para uma abordagem mais sofisticada, podemos combinar diferentes modelos em um mesmo modelo, o que é chamado de *ensemble* em aprendizado de máquina.

Aplicamos duas formas mais comuns de combinação de modelos, ***Bagging*** e ***Boosting***.

1. ***Bagging*** (com árvores de decisões hipotéticas):

```
#Arvores de decisao
arvore <-
  po("scale") %>>%
  po("learner", learner = lrn("classif.rpart", predict_type = "prob")) %>%
  as_learner()

#Bagging (sumsample + bootstrap)
bagging <-
  po("subsample", frac = 1, replace = T) %>>%
  po("learner", learner = lrn("classif.rpart")) %>%
  ppl("greplicate", ., 10) %>>%
  po("classifavg", innum = 10) %>%
  as_learner()

#Treinando as pipelines para a comparacao:
ensemble <- benchmark_grid(
  tasks = tsk,
  learners = list(arvore, bagging),
  resamplings = rsmp("holdout", ratio = 0.7))

resultadosensemble <- benchmark(ensemble)
```

```
resultadosensemble$score(msrs(c("classif.acc", "classif.bacc", "classif.ce")))
```

```
##                                   uhash nr              task
## 1: 19ec148e-772d-4478-93c7-d889dd633463  1 <TaskClassif[50]>
## 2: a19eab47-fda3-4744-903f-6f8b82831c8b  2 <TaskClassif[50]>
##                  task_id            learner
## 1: discursostreinobow$df <GraphLearner[38]>
## 2: discursostreinobow$df <GraphLearner[38]>
##
## 1:
## 2: subsample_1.subsample_2.subsample_3.subsample_4.subsample_5.subsample_6.subsample_7.subsample_8.su
##                 resampling resampling_id iteration              prediction
## 1: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
## 2: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
##    classif.acc classif.bacc classif.ce
## 1:   0.9315068    0.9322531 0.06849315
## 2:   0.9315068    0.9347222 0.06849315
```

Verificamos que o modelo de Random Forest ainda parece melhor que o ensemble de *Bagging*.

2. ***Extreme Gradient Boosting***:

```
#Extreme Gradient Boosting
gr_xgboost1 <-
  po("learner", learner = lrn("classif.xgboost", nrounds = 10, predict_type = "prob")) %>%
  as_learner()

gr_xgboost2 <-
```

```
  po("learner", learner = lrn("classif.xgboost", nrounds = 50, predict_type = "prob")) %>%
  as_learner()

gr_xgboost3 <-
  po("learner", learner = lrn("classif.xgboost", nrounds = 200, predict_type = "prob")) %>%
  as_learner()

modelosxgboost <- benchmark_grid(
  tasks = tsk,
  learners = list(gr_xgboost1, gr_xgboost2, gr_xgboost3),
  resamplings = rsmp("holdout", ratio = 0.7))

resultados <- benchmark(modelosxgboost)
```

```
resultados$score(msrs(c("classif.acc", "classif.bacc", "classif.ce")))
```

```
##                                   uhash nr              task
## 1: ed71b2d0-4914-4ffe-b4ca-ddecf659b784  1 <TaskClassif[50]>
## 2: b96110c2-2e00-4013-826f-6fb237d01818  2 <TaskClassif[50]>
## 3: 3baf2409-5332-41aa-aa9d-5431cb73f483  3 <TaskClassif[50]>
##                  task_id            learner      learner_id
## 1: discursostreinobow$df <GraphLearner[38]> classif.xgboost
## 2: discursostreinobow$df <GraphLearner[38]> classif.xgboost
## 3: discursostreinobow$df <GraphLearner[38]> classif.xgboost
##                resampling resampling_id iteration              prediction
## 1: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
## 2: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
## 3: <ResamplingHoldout[20]>       holdout         1 <PredictionClassif[20]>
##    classif.acc classif.bacc classif.ce
## 1:   0.9589041    0.9687500 0.04109589
## 2:   0.9726027    0.9791667 0.02739726
## 3:   0.9589041    0.9687500 0.04109589
```

Foi possível observar que o método do ***Boosting*** performa de maneira bastante similar ao do ***Random Forest***, com acurácia maior que 0,9. Eventuais diferenças podem ser ruídos decorrentes dos hiperparâmetros deste ensemble.

8. Predição

```
#Predicao Random Forest
modeloforest <- forest$train(tsk)
pred1 <- modeloforest$predict_newdata(discursostestebow)
pred1
```

```
#Confere validacao com metricas de teste
pred1$confusion
```

```
##         truth
## response Dilma Lula Temer
##    Dilma    38    3     0
```

```
##    Lula      0   42     1
##    Temer     0    1    20
```

```
pred1$score(msr("classif.acc"))
```

```
## classif.acc
##    0.952381
```

```
#Predicao Extreme Gradient Boosting
modelosxgboost <- gr_xgboost2$train(tsk)
pred2 <- modelosxgboost$predict_newdata(discursostestebow)
pred2
```

```
#Confere validacao com metricas de teste
pred2$confusion
```

```
##         truth
## response Dilma Lula Temer
##    Dilma    36    3     1
##    Lula      1   41     0
##    Temer     1    2    20
```

```
pred2 <- pred2$score(msr("classif.acc"))
pred2
```

```
## classif.acc
##   0.9238095
```

9. Predição de Random Forest e Extreme Gradient Boosting exportando probabilidades:

Para testarmos a predição através das probabilidades expostadas, utilizamos o tipo de predição "***prob***" e a métrica de validação ***logloss*** sob os mesmos modelos acima explorados.

```
forestprop <-
  po("learner", learner = lrn("classif.randomForest", ntree = 100, predict_type = "prob"))  %>%
  as_learner()

xgboostprop <-
  po("learner", learner = lrn("classif.xgboost", nrounds = 50, predict_type="prob"))  %>%
  as_learner()
```

```
#Predicao Random Forest (Prop)
modeloforestprop <- forestprop$train(tsk)
pred3 <- modeloforestprop$predict_newdata(discursostestebow)
pred3
```

```
# Confere validação com métricas de teste
pred3$confusion
```

```
##          truth
## response Dilma Lula Temer
##    Dilma    38    1     0
##    Lula      0   45     0
##    Temer     0    0    21
```

```
pred3 <- pred3$score(msr("classif.logloss"))
pred3
```

```
## classif.logloss
##       0.5744878
```

```
#Predicao Extreme Gradient Boosting (Prop)
modeloxgboostprop <- xgboostprop$train(tsk)
pred4 <- modeloxgboostprop$predict_newdata(discursostestebow)
pred4
```

```
# Confere validação com métricas de teste
pred4$confusion
```

```
##          truth
## response Dilma Lula Temer
##    Dilma    36    3     1
##    Lula      1   41     0
##    Temer     1    2    20
```

```
pred4 <- pred4$score(msr("classif.logloss"))
pred4
```

```
## classif.logloss
##       0.2102302
```

De acordo com a métrica da acurácia, o modelo ***Boosting*** tende a revelar proporções mais altas de probabilidade de acerto. A métrica do ***logloss*** é o inverso do logaritmo da função de probabilidade, de forma que valores menores de logloss significam maior probabilidade de acerto do modelo. Sendo assim, o método ***Boosting*** é superior ao ***Random Forest***, nesse caso em particular.

10. Predição para base de validação com discursos sem indicação de autoria

Testamos, por fim, a acurácia do Extreme Gradient Boost na amostra de validação:

```
#Adequo a base de validacao externa
discursosvalidacaobow <- validacao %>%
  corpus(text_field = "discurso") %>%
  tokens() %>%
  dfm() %>%
  dfm_match(featnames(discursostreinobow$bow)) %>%
  as.matrix() %>%
  as_tibble() %>%
  janitor::clean_names()
```

Table 1: Predição - Projeto 1 - Tamiris Burin

Id do discurso	Presidente previsto
1	Dilma
2	Dilma
3	Lula
4	Temer
5	Dilma
6	Lula
7	Dilma
8	Temer
9	Lula
10	Lula
11	Lula
12	Lula
13	Temer
14	Dilma
15	Dilma
16	Lula
17	Temer
18	Dilma
19	Lula
20	Dilma
21	Temer
22	Dilma
23	Lula
24	Temer
25	Temer

```
#Predicao com o modelo Xgboost por logloss proporcional
predfinal <- modeloxgboostprop$predict_newdata(discursosvalidacaobow)

predtamirisburin <- head(cbind(validacao$id, as.character(predfinal$response)), 25)

#Tabela de Predição da base de validação com discursos sem indicação de autoria
tabelapredtamirisburin <- predtamirisburin %>%
  kable(
    caption ="<b>Predição - Projeto 1 - Tamiris Burin</b>",
    col.names = c("Id do discurso","Presidente previsto")) %>%
  kable_styling("striped", full_width = F)
tabelapredtamirisburin
```