Projeto 1

Tamiris Burin

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

1. Base de dados

```
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

```
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",
    n=2,
    stem = TRUE) %>>%
  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()
```

Vale notar as funções de processamento testadas: 1. tolower: minimiza todos os termos para uma comparação facilitada. 2. remove_punct: remove a pontuação quando normalmente o contexto do corpus não é analisado. 3. remove_numbers: remove os numeros por nao serem relevantes tambem para a análise. 4. min_termfreq: estipula um valor mínimo de frequência do termo em todos os documentos. 5. stopwords_language = "pt": remove as palavras pequenas e genericas chamadas de 'stopwords', como 'o', 'a', 'e' etc. que nao diferenciam o conteúdo dos textos. É preciso usar o banco de dados de stopwords com a função get_stopwords() para a língua apropriada. 6. n = 2: a identificacao de ngrams trata de representar os pares fequentes de palavras do texto. Podemos especificar se procuramos por cunjuntos de 2 ou mais palavras e estes que serao analisadas depois como palavras unicas. 7. stem: remove diferentes derivacoes dos termos, retornando os termos em seus radicais. Muito util para neutralizar plurais e verbos conjugados. 8. scheme_df = 'inverse': Estipula um peso por termo calculando a frequencia inversa desse termo no documento.

3. Comparando Pré-Processamentos

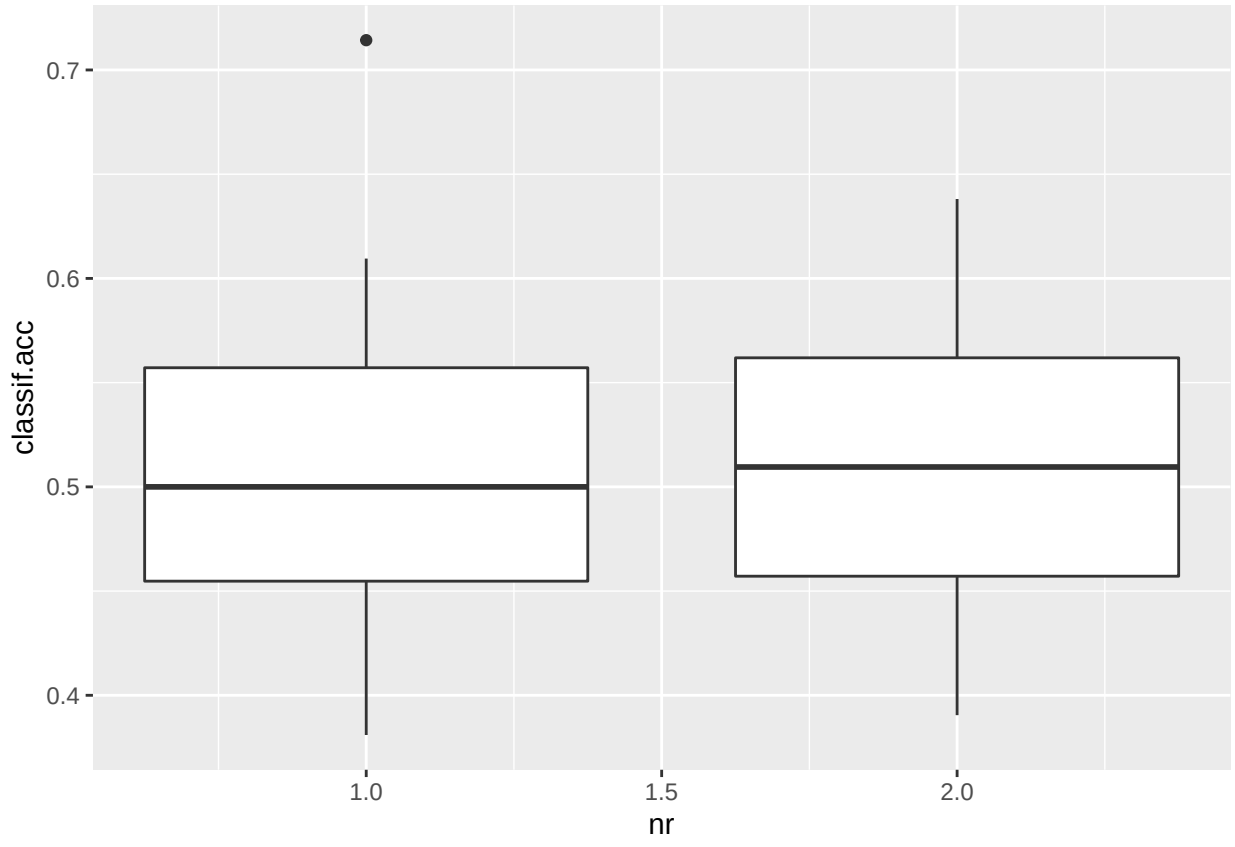
```
tsk <- as_task_classif(presidente ~ discurso, data = discursos)

  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 1 e 3 apresentaram os melhores resultados, vamos rodá-los 20 vezes cada um com 3 métricas de validações diferentes: Accuracy, Balanced Accuracy, e o Classification Error. Lembrando que quanto menor o índice do Classification Error, mais acurado é o resultado.

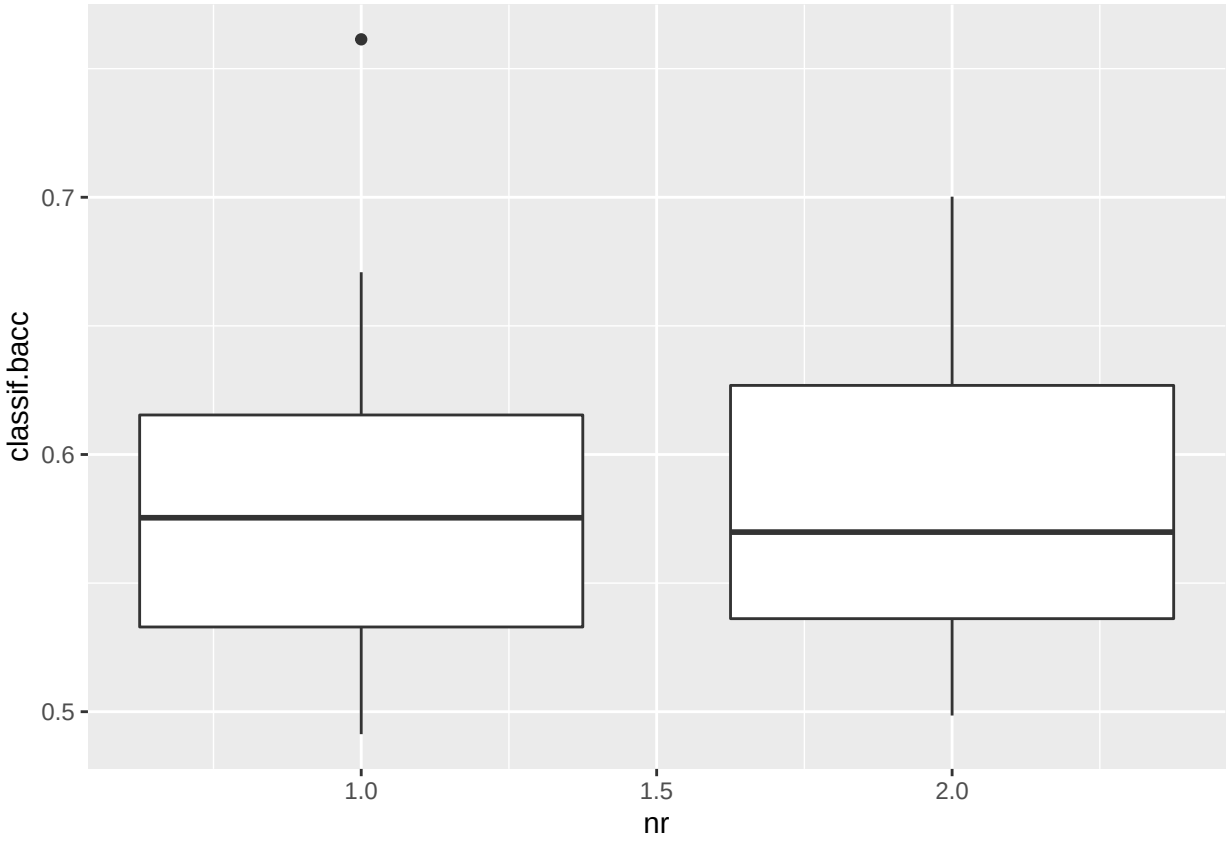
```
tsk <- as_task_classif(presidente ~ discurso, data = discursos)

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())
```

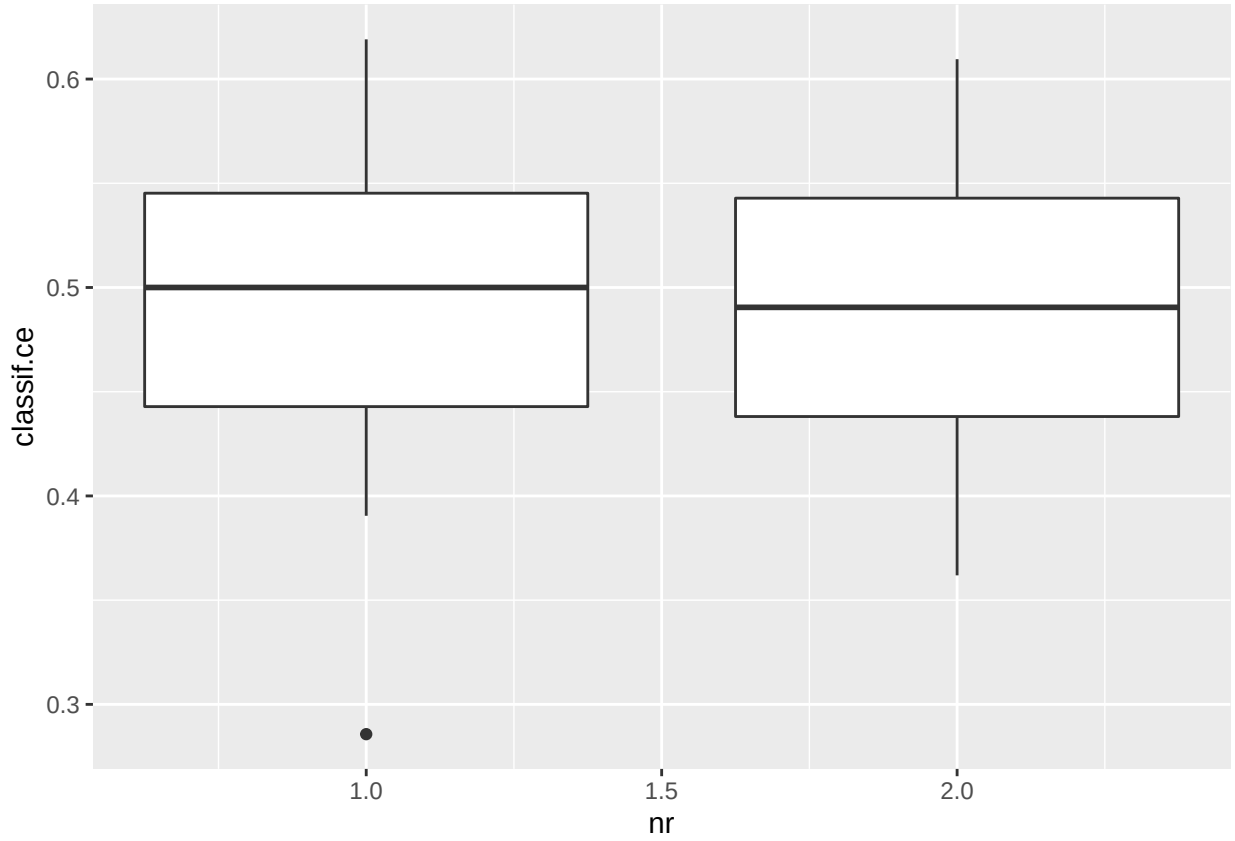
```
#Accuracy
resultados_processamentos %>%
  ggplot(aes(group = nr, y = classif.acc, x = nr)) +
  geom_boxplot()
```




```
#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

Aplicamos o pré-processamento n. 3:

```
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 Treino e Teste

```
#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)

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

6. Modelos

Aplicaremos três modelos:

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")))
```

Concluímos que o modelo Random Forest apresentou os melhores resultados.

7. Aplicando Ensembles

1. Aplicamos os ensembles Árvore de Decisão e Bagging:

```
#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")))
```

O modelo de Random Forest ainda parece melhor que os ensembles Árvore de Decisão e Bagging.

2. Aplicamos o 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")))
```

O modelo de Extreme Gradient Boosting performa de maneira similar ao 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    34    1     0
##    Lula      2   43     4
##    Temer     1    0    20
```

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

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

```
#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    34    0     0
##    Lula      3   44     1
##    Temer     0    0    23
```

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

```
## classif.acc
##   0.9619048
```

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

```
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    35    0     0
##    Lula      2   44     5
##    Temer     0    0    19
```

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

```
## classif.logloss
##       0.6176589
```

```
#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    34    0     0
##    Lula      3   44     1
##    Temer     0    0    23
```

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

```
## classif.logloss
##        0.163526
```

De acordo com a métrica da acurácia, o modelo Extreme Gradient Boost tende a revelar proporções mais altas de probabilidade de acerto. A métrica de validação 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 modelo de extreme gradient 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()

#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
```

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	Lula
19	Lula
20	Dilma
21	Temer
22	Dilma
23	Lula
24	Temer
25	Temer