Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

26 de dezembro de 2024

Lista 1

Prof. Drª. Thais Carvalho Rodrigues

Tópicos 2 - Aprendizagem de máquina

Aluno: Bruno Gondim Toledo | Matrícula: 15/0167636

1. Existem diferentes tipos de sistemas de aprendizado de máquina. Explique as classificações a seguir e cite 1 exemplo (algoritmo) de cada.

a) Aprendizado supervisionado ou não supervisionado

O aprendizado supervisionado é todo algoritmo que tem um label de referência, isto é, quando para as intâncias que desejamos treinar, testar e validar, sabemos o verdadeiro valor, que pode ser tanto numérico quanto categórico. Exemplos simples de aprendizado supervisionado seriam uma regressão linear para regressão e SVM (máquina de suporte vetorial) para classificação.

O aprendizado não supervisionado é aquele em que não temos um label de referência, e o algoritmo deve buscar padrões nos dados afim de, geralmente, classificá-los. Um exemplo rotineiro de algoritmo não supervisionado é o *k-means*.

b) Aprendizado baseado em instância ou baseado em modelo

O aprendizado baseado em modelo abarca todos os descritos acima, isto é, em que a partir de dados de treino, teste e validação iremos selecionar *features* e hiperparâmetros, por exemplo, para realiazar análise regressiva ou classificatória em dados posteriores sem *label*, ou para fazer previsões. Regressão linear, SVM e *k-means* por exemplo podem ser considerados aprendizado baseado em modelo.

O aprendizado baseado em instância é aquele em que não é construído um modelo, isto é, quando estamos apenas interessados, ou só é possível, descrever semelhanças e dissemelhanças na amostra obtida. Um algoritmo popular de aprendizado em instância é o KNN (*k-nearest neighborhood*)

c) Aprendizado online ou em batch

O aprendizado online ou em batch é aquele em que ou existe um fluxo constante de dados, ou a quantidade de dados é tão massiva que não é possível considerar todo o contexto para modelagem num tempo viável. Diversos algoritmos modernos tem essa especificidade, por conta da natureza da velocidade da informação atual. Boa parte deles é baseado em alguma variante do algoritmo Gradiente Descendente, que permite encontrar mínimos de funções que otimizem um problema.

2. Comente 1 contexto no qual seria vantajoso utilizar um sistema de aprendizado de máquina e outro contexto no qual seria desvantajoso.

O aprendizado de máquina é crucial no nosso estilo de vida atual, em que somos constantemente bombardeados de informação — e em que os remetente desta informação tem um custo associado a este bombardeamento. Portanto, tanto para quem recebe como para quem envia a informação (seja um e-mail, uma propaganda) irá buscar filtrar receber apenas aquilo que o interessa, e enviar apenas para o público alvo que compense este custo. Logo, tarefas como separação de e-mail em spam e não spam; separação mecânica de grãos (café, amendoim, etc), público alvo de tipos específicos de publicidade e propaganda em geral são aplicações valorosas e pouco nocivas do aprendizado de máquina.

Por outro lado, o surgimento desta tecnologia tão valorosa e útil também carrega uma quantidade significante de misticismo e problemáticas, em que pessoas pouco versadas podem acreditar que um julgamento maquínico é capaz de substituir o trabalho humano. Esta é uma visão equivocada por principalmente dois motivos. Primeiro, que a máquina não é capaz de fazer tudo, enfrentando limitações para algumas tarefas e tendo um custo associado de manutenção, energia e infraestrutura que podem não ser simples ou viáveis de implementar. E segundo, por enfrentar paradigmas éticos sociais em determinados contextos, visto que em geral os dados disponíveis refletem o verdadeiro parâmetro, isto é, reflete padrões sociais deletérios existentes como racismo, machismo e xenofobia. Ver SILVA (2022) e BOLUKBASI (2016).

3. Qual é o propósito do conjunto de dados de treino, validação e teste?

Treino

Os dados de treino são o que utilizaremos de fato para ajustar o modelo, que serão os dados inseridos para escolha de *features* e fornecerão parâmetros do modelo. Em geral, utilizaremos a maior parte dos dados disponíveis nesta etapa (em torno de 70%).

Validação

Os dados de validação serão os dados em que utilizaremos o ajuste do modelo para fazer ajustes de hiperparâmetros. Esta é uma etapa intermediária, em que o modelo terá oportunidade de ser testado em dados ainda não visto, para que possamos verificar se alguns dos parâmetros, features e hiperparâmetros selecionados precisam de algum tipo de ajuste, afim de eliminar viéses do conjunto de treino. Em geral utilizaremos em torno de 15% dos dados nesta etapa.

Teste

Esta é a etapa final de validação do modelo. Com as *features* e hiperparâmetros definidos, iremos testar o quão bem o nosso modelo performa sobre dados nunca antes vistos. Nesta etapa iremos determinar se conseguimos chegar a um ajuste decente, ou se um possível indicativo de bom ajuste não confirmado se trata apenas de *overfitting* sobre os conjuntos anteriores. Em geral, utilizaremos de 15 a 30% dos dados nesta etapa.

4. Caso o modelo tenha um bom desempenho nos dados de treinamento, mas não nos dados de validação, o que está acontecendo? Como corrijir esse problema?

Neste caso, quase certamente o modelo está sofrendo *overfitting*, isto é, está descrevendo perfeitamente o conjunto de treino, mas falhando em explicar dados ainda não vistos. Em geral isto se dá com o refino extremo do modelo, adicionando muitas dimensões, *features* e minimizando o erro à quase zero. Neste caso, o ideal é reiniciar todo o procedimento de modelagem, re-escolhendo *features* e hiperparâmetros mais "folgados", isto é, que acomodem com relativo êxito os conjuntos de treino, mas deixando um pouco da variabilidade natural dos dados existir sem ajuste.

5. Qual é a diferença entre parâmetro e hiperparâmetro?

No paradigma frequentista, poderia se dizer que um parâmetro é uma característica verdadeira, inata e imutável de uma determinada população; como uma lei que rege a distribuição de probabilidade desta população. Em geral, o parâmetro é um valor numérico, associado a uma distribuição de probabilidade, e assume valores de acordo com o suporte de cada distribuição, conforme sua função de distribuição de probabilidade. Um hiperparâmetro é muito mais uma característica do modelo. Isto é, em geral os algorítmos de aprendizado maquínico contém valores *Setups* que devem ser informados pelo usuário, como número de *clusters* em um algoritmo *k-means*, número de vizinhanças num algoritmo *KNN*, *threshold* num *SVM* e assim por diante. Este valor em geral está associado a natureza dos dados em questão, visto que estes são modelos gerais, isto é, são utilizados para descrever e modelar dados de diversas naturezas, cada um com particularidades que devem ser indiretamente informados pelo usuário ao modelo através dos hiperparâmetros. Por exemplo, num exemplo classificatório em que o usuário sabe que existem 4 classes possíveis para os dados, este é um dado a ser informado a um algoritmo como *k-means*, se tratando portanto de um hiperparâmetro do mesmo.

6. Como escolher o hiperparâmetro k do algoritmo KNN?

Esta é uma decisão do analista que deverá ser feita, em geral de acordo com a natureza dos dados em questão. Em geral, valores muito baixos (k=1) tendem a viesar os resultados, enquanto valores muito altos ($\lim_{k\to\infty}$) tendem a só retornar uma das classes para todos os dados, não fazendo nenhum tipo de separação. Portanto, é importante utilizar o conjunto de validação para seleção do melhor hiperparâmetro neste caso. Para isto, podemos por exemplo fazer uma análise gráfica, plotando os erros de predição nos conjuntos de treino e validação contra diversos valores de *k*, afim de determinar o valor ótimo que leve a um bom ajuste/previsão, tanto para dados de treinamento quanto para dados de teste. Em geral, estes gráficos terão a forma de uma "trombeta invertida", isto é, para pequenos valores de k, ajustando perfeitamente ao treino e nada na validação, e aproximando a quantidade de erros em ambos os conjuntos para maiores valores de K, até um *thresold* de divergência em ambos os casos, para quando o valor *k* é demasiado alto.

7. É necessário escalonar as covariáveis para aplicação do algoritmo KNN? Por quê? E na floresta aleatória?

Não e não. Na realidade, depende da natureza dos dados. O algoritmo KNN é especialmente sensível a covariáveis com valores em absoluto muito elevados, podendo dar demais importância a uma covariável em que, se escalonada, a importância relativa não seria tão grande. Portanto, se estamos trabalhando com unidades de medidas muito díspares, é essencial para o algoritmo KNN o escalonamento das covariáveis. Entretanto, é importante saber quando e porquê se está fazendo uma transformação na(s) covariável(is), e não simplesmente "copiar e colar" um código *scale()* de forma acrítica, em situações em que isto sequer seria necessário e possa levar a uma dificuldade na interpretação dos *outputs* desnecessariamente. Claro que isto é apenas um preciosismo teórico, pois teremos **sim** que escalonar as covariáveis em 99,99% das situações em que utilizamos o KNN!

No caso da floresta aleatória, praticamente nunca é necessário o escalonamento das covariáveis. Diferentemente do KNN, as florestas aleatórias são muito menos propensas à dar demasiada importância a uma *feature* particular, também por que durante o treinamento deste modelo, as covariáveis não são inseridas em todas as árvores construídas, justamente evitando este tipo de problema.

8. Ajuste o KNN no conjunto de dados MNIST utilizando 2 medidas de similaridade diferentes e mensure o desempenho dos modelos.

Buscando os pacotes e dados

```
if (!require("pacman")) install.packages("pacman")
p_load(keras,tidymodels,kknn)
tidymodels_prefer()
mnist <- dataset_mnist()
```

Preparando os dados

```
train <- mnist$train$x
train_labels <- mnist$train$y

train <- array_reshape(train, c(nrow(train), 784)) / 255
train <- data.frame(train, label = train_labels)

test <- mnist$test$x
test_labels <- mnist$test$y

test <- array_reshape(test, c(nrow(test), 784)) / 255
test <- data.frame(test, label = test_labels)

train$label <- as.factor(train$label)
test$label <- as.factor(test$label)
```

Acima, foi normalizado os valores dos pixels além de ser alterada a geometria dos dados “crus”. Pela natureza um tanto estranha dos dados, diferente do que se convenciona tratar em R, fiz as transformações diretamente ao invés de utilizar o *::recipes*, pois seria um tanto confuso para mim neste caso.

Criarei também um conjunto de validação

```
set.seed(150167636)
data <- initial_split(train, prop = 0.8)
train <- training(data)
validation <- testing(data)
```

Criando o modelo utilizando o *tidymodels*

Definindo o modelo

```
knn_model = nearest_neighbor(
  mode = "classification",
  engine = "kknn",
  neighbors = 11,
  dist_power = 2
)
```

Definindo o workflow

```
knn_workflow <- workflow() %>%
  add_model(knn_model) %>%
  add_formula(label ~ .)
```

Finalmente, ajustando o modelo

```
rm(data,mnist,train_labels,test_labels)
knn_fit <- fit(knn_workflow, data = train)
```

Aqui, pode-se validar os hiperparâmetros do modelo

```
validacao <- predict(knn_fit, new_data = validation) %>%
  bind_cols(validation) %>%
  select(.pred_class,label) %>%
  mutate(label = factor(label)) %>%
  metrics(truth = label, estimate = .pred_class)
```

```
kable(validacao)
```

.metric	.estimator	.estimate
accuracy	multiclass	0.9515833
kap	multiclass	0.9461747

Com os hiperparâmetros inicialmente chutados (k=11, restante = *default*), o modelo parece já não precisar de ajustes de hiperparâmetros.

Por fim, o teste do modelo sob os dados de teste.

```
resultados <- predict(knn_fit, new_data = test) %>%
  bind_cols(test) %>%
  select(.pred_class,label) %>%
  mutate(label = factor(label)) %>%
  metrics(truth = label, estimate = .pred_class)
```

```
kable(resultados)
```

.metric	.estimator	.estimate
accuracy	multiclass	0.9465000
kap	multiclass	0.9405275

No que pese ter demorado quase 2 horas entre ajuste, validação e teste, o modelo performou muito bem neste caso, chegando a uma acurácia de mais de 94% sobre o conjunto de teste.

Testando o segundo exemplo solicitado:

Como foi solicitado para mudar a métrica de distância apenas, irei manter o hiperparâmetro k = 11, que performou bem no último modelo. No modelo anterior, não alterei o parâmetro de distância do modelo, deixando *default* p = 2. O modelo utiliza a definição de distância de Minkowsky, que generaliza diversas métricas de distância. Nesta, para p = 2, temos a distância Euclidiana. Portanto, para este segundo modelo utilizarei p = 1, em que temos a distância de Manhattan.

OBS: Tentei várias vezes deixar por mais de 8 horas rodando, e não obtive resultados. Portanto, tive de limitar o tamanho dos dados.

```
knn_model2 = nearest_neighbor(
  mode = "classification",
  engine = "kknn",
  neighbors = 11,
  dist_power = 1
)

knn_workflow2 <- workflow() %>%
  add_model(knn_model2) %>%
  add_formula(label ~ .)

train_part = sample_n(train,1000)
test_part = sample_n(test,1000)

knn_fit2 <- fit(knn_workflow2, data = train_part)

resultados2 <- predict(knn_fit2, new_data = test_part) %>%
  bind_cols(test_part) %>%
  select(.pred_class,label) %>%
  mutate(label = factor(label)) %>%
  metrics(truth = label, estimate = .pred_class)
```

```
kable(resultados2)
```

.metric	.estimator	.estimate
accuracy	multiclass	0.8670000
kap	multiclass	0.8518713

Comparando os resultados

Resultados do primeiro modelo

```
kable(resultados)
```

.metric	.estimator	.estimate
accuracy	multiclass	0.9465000
kap	multiclass	0.9405275

Resultados do segundo modelo

```
kable(resultados2)
```

.metric	.estimator	.estimate
accuracy	multiclass	0.8670000
kap	multiclass	0.8518713

É difícil comparar este rendimento. Em números absolutos, o primeiro modelo performou melhor — porém o alto tempo de execução do algoritmo inviabiliza um ajuste de hiperparâmetros. Também, o primeiro modelo foi treinado com 48.000 instâncias de treinamento, ao passo que o segundo recebeu um conjunto de apenas 1.000 instâncias afim de tornar razoável o tempo de execução.

É importante observar estas dificuldades, visto que no curso estamos acostumados a trabalhar com conjuntos de dados pequenos, com algumas poucas centenas de observações, e tudo funciona perfeitamente. Porém, ao escalonar esta dimensão exponencialmente, vemos que temos que nos atentar à isto também na hora de selecionar um modelo e estratégias para lidar com ele.

9. Explique a medida de desempenho AUC (área sob a curva ROC).

O desempenho AUC é uma das formas de avaliação de modelo, talvez uma das que traga de forma mais visual e clara os resultados objetivos de uma modelagem. Na curva ROC, teremos uma espécie de hipérbole formada sobre a seção noroeste de um plano cortado por uma semireta de 45º. Estamos interessados em analisar a área preenchida pela curva hiperbólica no plano, isto é, quão mais próximo de meio do plano noroeste contido sob a curva, melhor será o resultado do modelo. Neste, iremos plotar os falsos positivos contra os verdadeiros positivos, isto é, (1 - especificidade) contra a sensibilidade. Quão mais próxima a curva ROC estiver da semireta 45º, mais indica que o modelo não se sobrepõe ao acaso. Quão mais próxima do vértice noroeste do plano, mais indica que o modelo está realizando uma boa rotina preditiva.

10. Como a árvore de decisão escolhe a regra de decisão de 1 nó (covariável e valor de partição)?

A decisão será pelo ganho de entropia, isto é, o algoritmo irá testar divisões possíveis, e calcular dentre as divisões possíveis aquela que maximiza a entropia (informação) dos dados.

11. Compare o KNN e a árvore de decisão (vantagens/desvantagens).

O KNN é um algoritmo extremamente simples, fácil de descrever e analisar a performance do modelo, e de explicabilidade ao usuário do porquê uma determinada decisão ou agrupamento fora formado. Ainda, consegue performar relativamente bem ante a algoritmos extremamente sofisticados em algumas situações, sendo um modelo útil para alguns cenários. Como visto anteriormente, um destaque negativo do KNN é a performance extremamente lenta para conjuntos de dados grandes.

A árvore de decisão é uma modelagem interessante por ter um viés absolutamente computacional. A quantidade de cálculos envolvidas inviabilizaria completamente realizar uma análise sem apoio computacional, e as decisões formadas pelo modelo podem não ser muito intuitivas ou fáceis de explicar ao usuário. Entretanto, é uma modelagem extremamente poderosa e robusta, que tem a capacidade para ser expandida ainda para modelos de *random forest* e *ensemble* (*boost*) por exemplo, com testes indiando ser uma modelagem extremamente robusta ante a *outliers* e viéses.

Num geral, espera-se que modelos baseados em árvores sejam ao menos tão bons quanto modelos KNN, e dada a facilidade de acesso a um usuário simples a poderes computacionais gigantescos (ex. eu, um simples estudante dispondo de uma máquina de 16 núcleos de processamento e 20Gb de memória RAM), contar com ferramentas de modelagem computacionalmente dependentes não é mais um desafio, mas sim uma solução.

12. O que fazer se uma árvore de decisão estiver sobreajustando aos dados de treino?

Na prática, partir para uma modelagem mais robusta como *random forest*, que trás a mesma filosofia da árvore de decisão porém mais robusta a *overfitting*. Entretanto, para ficar com uma árvore apenas, podemos pensar em seleção de *features*, escalonamento de covariáveis, ajustes de hiperparâmetros e demais rotinas comuns para aprendizagem de máquina, como utilizar técnicas de reamostragem bootstrap, tratamento ou remoção de *outliers*, etc.

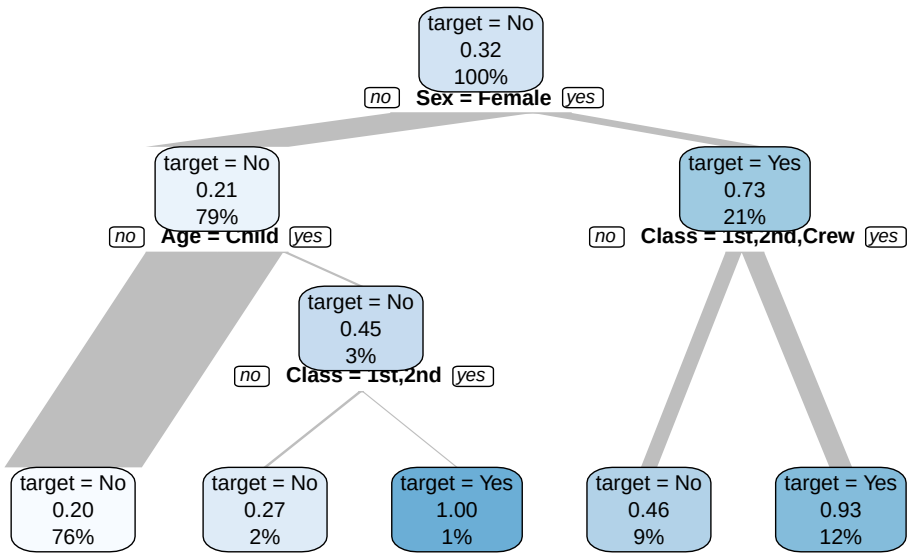
13. Ajuste uma árvore de decisão no conjunto de dados Titanic para prever se o passageiro sobreviveu ou não ao acidente. Ajuste a profundidade da árvore por meio de um gridsearch no conjunto de validação. Avalie o desempenho do modelo.

Os dados:

```
p_load(explore)
data <- use_data_titanic()
```

Existem diversos conjuntos de dados Titanic. Este está categorizado, isto é, não contém as informações específicas de idade e nem de preço da passagem.

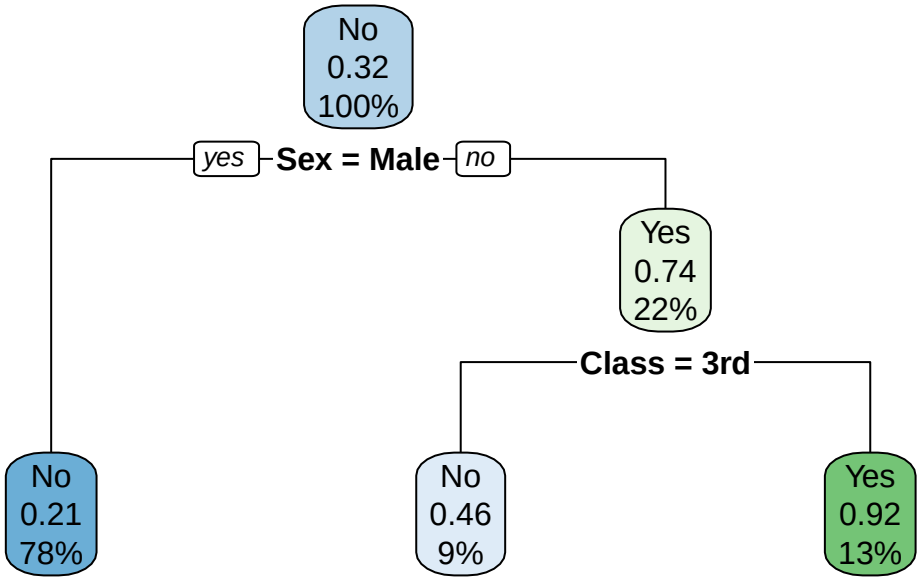
```
data = data |>
  mutate(Class = factor(Class),
         Sex = factor(Sex),
         Age = factor(Age),
         Survived = factor(Survived))
data %>% explain_tree(target = Survived)
```



Fazendo uma “modelagem exploratória” para observar o conteúdo dos dados, vemos que aparenta ser possível realizar uma modelagem do tipo árvore de decisão para este conjunto de dados. Faremos uma modelagem de árvore propriamente dita agora, com as etapas necessárias para a construção do modelo.

```
p_load(tidymodels)
tidymodels_prefer()

split <- initial_validation_split(data, prop = c(0.7, 0.15), strata = "Survived")
train <- training(split)
val <- validation(split)
test <- testing(split)

model <- decision_tree(mode = "classification") %>%
  set_engine("rpart")

recipe <- recipe(Survived ~ ., data = train)

wf <- workflow(recipe,model)

fit <- fit(wf,train)

rpart.plot::rpart.plot(fit$fit$fit$fit)
```

```
Warning: Cannot retrieve the data used to build the model (model.frame: objeto '..y' não e
To silence this warning:
    Call rpart.plot with roundint=FALSE,
    or rebuild the rpart model with model=TRUE.
```




```
val_res <- predict(fit, new_data = val %>% select(-Survived))
val_res <- bind_cols(val_res, val %>% select(Survived))

table(val_res)
```

```
           Survived
.pred_class  No Yes
        No  223  66
        Yes   3  38
```

Fazendo agora com tuning de parâmetros

```
model <- decision_tree(
  mode = "classification",
  tree_depth = tune()) %>%
  set_engine("rpart")

recipe <- recipe(Survived ~ ., data = train)

wf <- workflow() %>%
  add_recipe(recipe) %>%
  add_model(model)

grid <- grid_regular(tree_depth(),
                             levels = 15)

set.seed(150167636)
```

```
folds <- vfold_cv(val)

tuning <- wf %>%
  tune_grid(resamples = folds,
            grid = grid)

# tuning %>%
#   collect_metrics()

tuning %>%
  show_best(metric = "accuracy")
```

```
# A tibble: 5 x 7
  tree_depth .metric  .estimator  mean     n std_err .config
       <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1          1 accuracy binary     0.797    10  0.0169 Preprocessor1_Model01
2          2 accuracy binary     0.797    10  0.0169 Preprocessor1_Model02
3          3 accuracy binary     0.797    10  0.0169 Preprocessor1_Model03
4          4 accuracy binary     0.797    10  0.0169 Preprocessor1_Model04
5          5 accuracy binary     0.797    10  0.0169 Preprocessor1_Model05
```

Vemos que o gridsearch indica que qualquer profundidade entrega a mesma acurária, o que é um resultado que não faz sentido algum, ao menos que fosse um conjunto de dados perfeitamente separado com acurácia = 1, que não é o caso.

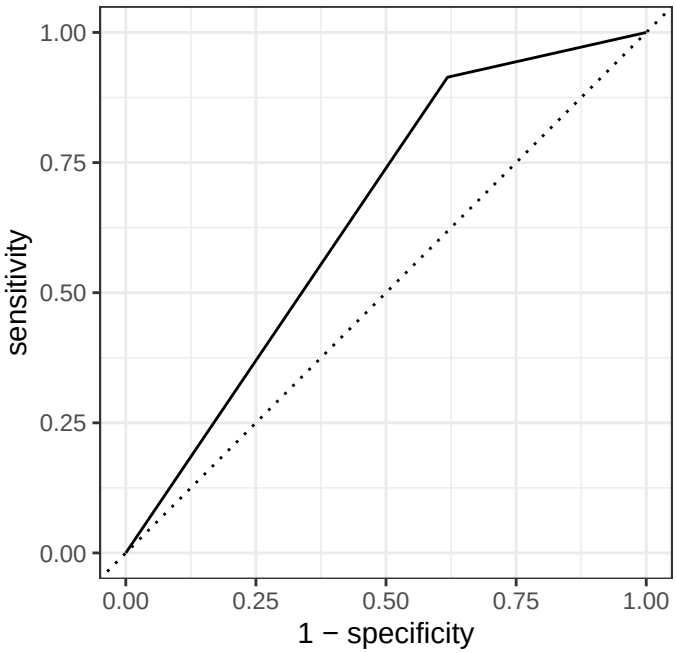
```
best_tree <- tuning %>%
  select_best(metric = "accuracy")
```

```
final_wf <-
  wf %>%
  finalize_workflow(best_tree)
```

```
final_fit <-
  final_wf %>%
  last_fit(split)

final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.737 Preprocessor1_Model1
2 roc_auc     binary         0.648 Preprocessor1_Model1
3 brier_class binary         0.195 Preprocessor1_Model1
```

```
final_fit %>%
  collect_predictions() %>%
  roc_curve(truth=Survived, .pred_No) %>%
  autoplot()
```



Vemos que este modelo performou significativamente mal pela curva ROC, resultado de algum erro na etapa de tuning do hiperparâmetro — o que é um resultado estranho, visto que segui a risca o passo a passo do livro https://www.tidymodels.org/start/tuning/

Referências

SILVA, T. Racismo algorítmico: inteligência artificial e discriminação nas redes digitais. (2022)

BOLUKBASI, T. et al. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. 2016.

https://www.tidymodels.org/start/tuning/