Melhorando a Performance dos Modelos

ESTAT0109 – Mineração de Dados em Estatística

Prof. Dr. Sadraque E. F. Lucena

sadraquelucena@academico.ufs.br

http://sadraquelucena.github.io/mineracao

Objetivo da Aula

- Apresentar métodos de aprendizagem em conjunto (bagging e boosting) e ensinar como ajustar seus hiperparâmetros para melhorar o desempenho preditivo dos modelos.

Aprendizagem em Conjunto (Ensemble Learning)

- Métodos de aprendizado em conjunto são técnicas utilizadas para combinar múltiplos algoritmos de aprendizado a fim de obter um desempenho preditivo melhor do que poderia ser obtido por qualquer um dos algoritmos de aprendizado individualmente.
- Veremos dois tipos:
  - *Bagging*
  - *Boosting*

Bagging

Aprendizagem em Conjunto (Ensemble Learning)

Bagging

- *Bagging* vem de *Bootstrap Aggregating* (agregação *bootstrap*).
- A ideia é treinar vários modelos com diferentes subconjuntos dos dados de treinamento selecionados aleatoriamente e combinar suas predições para fazer a predição final.
  - Esta estratégia reduz a variâcia da predição, pois cada modelo é treinado com um conjunto levemente diferente de dados.
  - Ela é particularmente útil quando os modelos tendem a sobreajustar aos dados.
- Dois algoritmos de Bagging são comumente usados:
  - *Vanilla Bagging*
  - *Random Forest* (Floresta Aleatória)

Aprendizagem em Conjunto (Ensemble Learning)

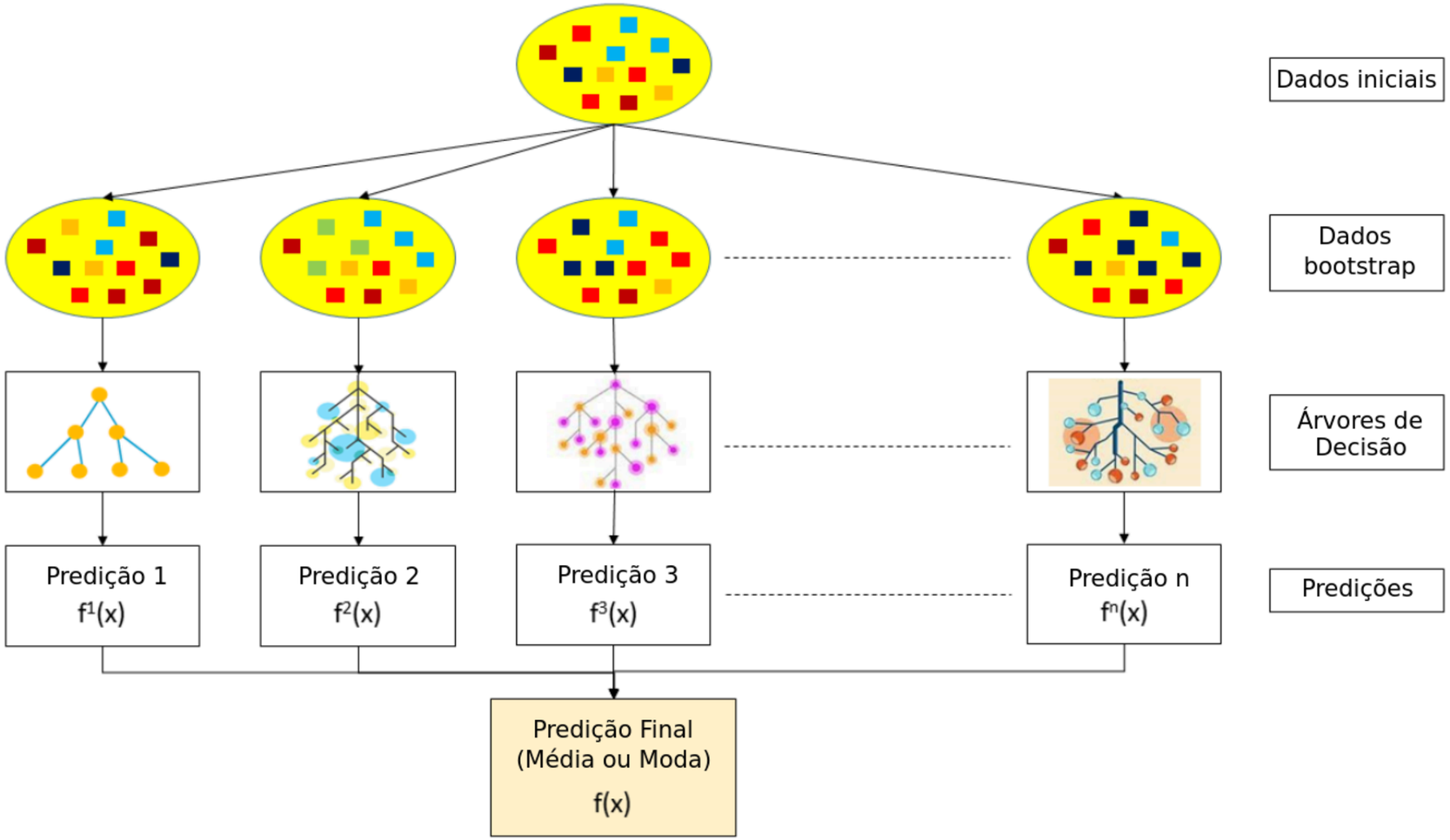
Vanilla Bagging

1. Obtemos amostras *bootstrap*: selecionamos de forma aleatória e com reposição instâncias do conjunto de dados, gerando subconjuntos de mesmo tamanho do original.
2. Usamos algum método (árvores de decisão, redes neurais, SVMs…) para predizer as respostas em cada amostra *bootstrap*.
3. Combinamos as saídas para uma resposta final.

- Em **problemas de regressão** a saída é a média das predições de cada modelo.
- Em **problemas de classificação** a saída é o voto da maioria dos modelos.
- O número de subconjuntos (amostras) N é um hiperparâmetro que pode ser ajustado.
- Exemplo: algoritmo `Bagged CART`.

Aprendizagem em Conjunto (Ensemble Learning)

Vanilla Bagging



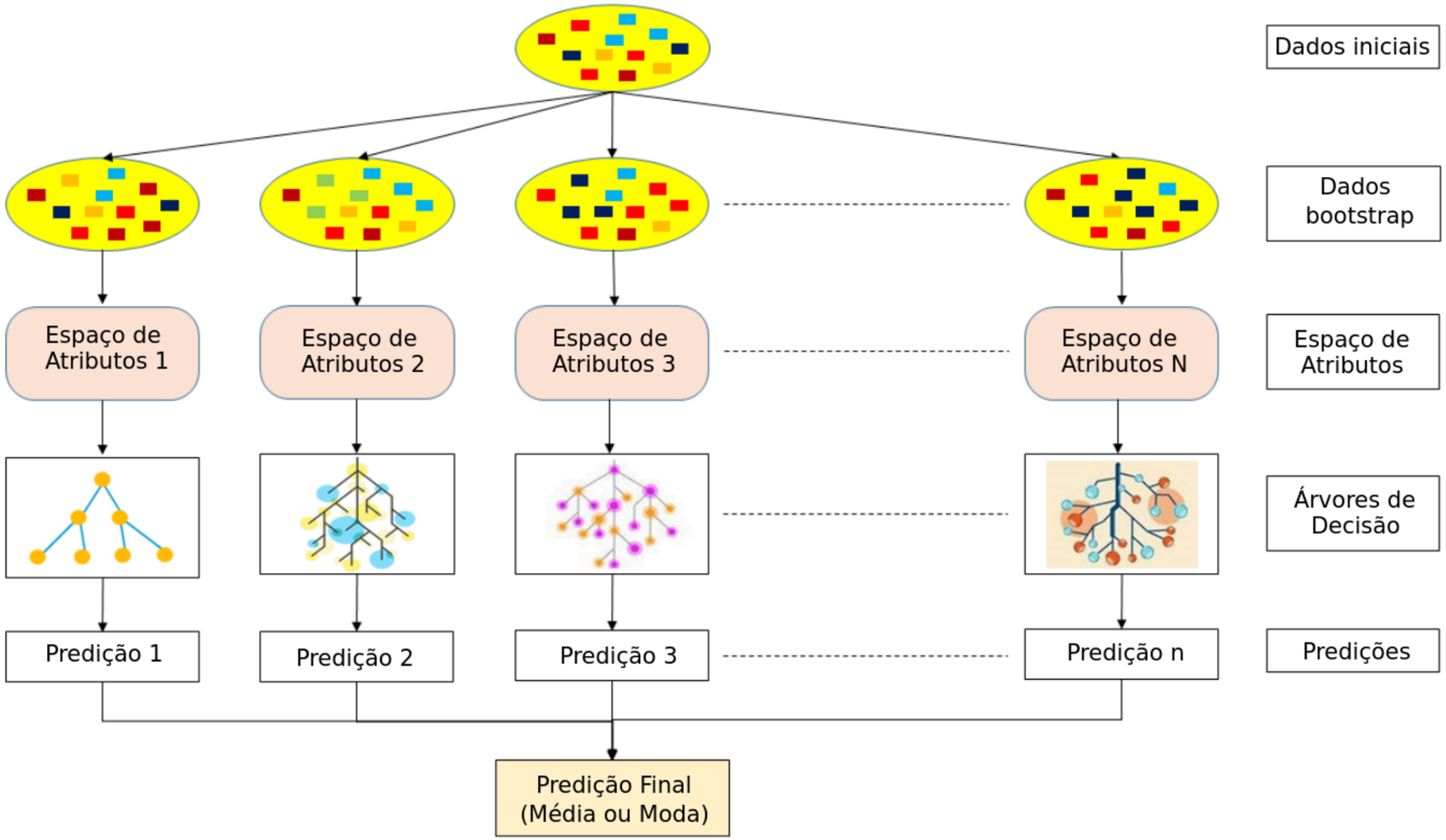
Aprendizagem em Conjunto (Ensemble Learning)

Random Forest

- É uma extensão do algoritmo *vanilla bagging*. O procedimento é:
  1. Obtenha N amostras *bootstrap*, selecionando aleatoriamente instâncias do conjunto de dados original, e também escolhendo alguns atributos de forma aleatória.
  2. Em cada amostra ajuste uma árvore de decisão para fazer previsões.
  3. Combine as previsões de todas as árvores para obter uma resposta final.
- A principal diferença do *random forest* para o *vanilla bagging* é que, além das instâncias, os atributos de cada subconjunto também são selecionados aleatoriamente.
- O número de atributos a serem considerados em cada subconjunto é um hiperparâmetro.

Aprendizagem em Conjunto (Ensemble Learning)

Random Forest



Aprendizagem em Conjunto (Ensemble Learning)

Random Forest

- **Vantagens**
  - Reduz risco de *overfitting*, pois ajusta modelos com observações e atributos levemente diferentes.
  - Maior acurácia.
  - Robustez a ruídos e valores ausentes.
- **Desvantagens**
  - Demanda tempo e poder computacional, pois ajusta várias árvores.
  - Interpretabilidade mais difícil que em uma árvore de decisão.

Boosting

Boosting

- O método *Boosting* combina vários aprendizes fracos para formar um aprendiz forte.
- Procedimento:
  1. Inicialmente um modelo é treinado e avaliado o erro de predição em cada instância.
  2. Um segundo modelo é ajustado, dando mais atenção às observações com previsões incorretas. Isso é feito sucessivamente (vezes) para reduzir o erro de predição.
  3. A saída é a combinação dos resultados dos modelos.
- Três algoritmos populares de *boosting* são `AdaBoost`, `Gradient Boosting` e `XGBoost`.
- O algoritmo `C5.0` também permite fazer *boosting*.
- O número de árvores de decisões usadas é um hiperparâmetro que pode ser ajustado.

N

N

Boosting

- Vejamos melhor a ideia do algoritmo usando `Gradient Boosting` para regressão:
  1. Treine Árvore-1 aos dados e estime .
  2. Calcule os resíduos a partir da Árvore-1. Treine Árvore-2 usando como variável resposta os resíduos .
  3. Repita o processo usando os resíduos da árvore ajustada anteriormente como variável resposta até que árvores sejam treinadas.
  4. Combina as predições para obter a predição final.
- A predição de será a combinação das predições de cada árvore:

$\hat{y}_i$

r_{1i}

r_{1i}

N

y_i

$$y_{pred} = \hat{y}_i + \eta\,\hat{r}_{1i} + \eta\,\hat{r}_{2i} + \eta\,\hat{r}_{Ni}$$

Boosting

Exemplo

Suponha que queremos estimar o preço de uma casa baseado na idade, metragem quadrada e localização.

Age	Square Footage	Location	Price
5	1500	5	480
11	2030	12	1090
14	1442	6	350
8	2501	4	1310
12	1300	9	400

- **Passo 1**: uma árvore de decisão é treinada e preditos os valores das casas.

Boosting

Exemplo

- **Passo 2**: Treinamos uma segunda árvore de decisão considerando como reposta os resíduos da árvore anterior, .
- O resíduo é calculado: r_{1i}

r_{1i} r_{1i}

$$r_{1i} = y_i - \hat{y}_i$$

Age	Square Footage	Location	Price	Predicted $\hat{y}_1$	Residuals r_1
5	1500	5	480	440	40
11	2030	12	1090	880	210
14	1442	6	350	400	-50
8	2501	4	1310	1010	300
12	1300	9	400	600	-200

Boosting

Exemplo

- **Passo 3**: Os resíduos dessa nova árvore são obtidos e usados como resposta no treino da árvore seguinte.

Age	Square Footage	Location	Price	Predicted $\hat{y}_1$	Residuals r_1	Predicted $\hat{y}_2$	Residuals r_2
5	1500	5	480	440	40	20	20
11	2030	12	1090	880	210	200	10
14	1442	6	350	400	-50	0	-50
8	2501	4	1310	1010	300	240	60
12	1300	9	400	600	-200	-50	-150

Boosting

Exemplo

- **Passo 4**: O processo continua até que N árvores sejam treinadas.
- A predição final é a soma das predições de todas as árvores ponderadas pela taxa de aprendizado.

Age	Square Footage	Location	Price	Predicted $\hat{y}_1$	Residuals r_1	Residuals r_2	...	Price Prediction y_{pred}
5	1500	5	480	440	40	20		478
11	2030	12	1090	880	210	10		1091
14	1442	6	350	400	-50	-50		335
8	2501	4	1310	1010	300	60		1311
12	1300	9	400	600	-200	-150		400

Ajuste de Parâmetros

Ajuste de Parâmetros

- Os parâmetros que controlam o comportamento e o desempenho dos algoritmos de aprendizagem de máquina são comumente chamados de *hiperparâmetros*.
- Durante o processo de treinamento de um modelo de aprendizagem de máquina, é essencial ajustar esses hiperparâmetros para melhorar o desempenho preditivo do modelo.
- De acordo com o método utilizado, diferentes hiperparâmetros podem ser ajustados.

Ajuste de Parâmetros

- Exemplos:

Método	Tipo de aprendizado	Função no R	Parâmetros
k-NN	Ambos	`knn`	`k`
Naive Bayes	Classificação	`nb`	`fL`, `usekernel`
Árvore C5.0	Classificação	`C5.0`	`model`, `trials`, `winnow`
Árvore CART	Ambos	`rpart`	`cp`
Árvore XGBoost	Ambos	`xgbTree`	`nrounds`, `max_depth`, `eta`, `gamma`, `colsample_bytree`, `min_child_weight`, `subsample`

Ajuste de Parâmetros

- k-Nearest Neighbors:
  - `k`: número de vizinhos mais próximos.
- Naive Bayes:
  - `fL`: incorpora o suavizador de Laplace, usado para lidar com a probabilidade zero de certos eventos ocorrerem em dados de treinamento;
  - `usekernel`: especifica se o método deve usar uma estimativa de densidade de kernel (`TRUE`) para variáveis contínuas em vez de uma estimativa gaussiana padrão (`FALSE`).

Ajuste de Parâmetros

- Árvore C5.0:
  - `model`: se será uma árvore (`tree`) ou conjunto de regras (`rules`).
  - `trials`: número de iterações boost.
  - `winnow`: usa todos os atributos (`FALSE`) ou apenas os mais importantes (`TRUE`).
- Árvore CART:
  - `cp`: parâmetro de complexidade. Valor de 0 a 1 usado para podar a árvore (quanto menor o valor de `cp`, maior a árvore).
- Random Forest:
  - `mtry`: número de atributos que serão selecionados aleatoriamente em cada árvore.

Ajuste de Parâmetros

- Árvore XGBoost:
  - `nrounds`: número de iterações de boosting.
  - `max_depth`: profundidade máxima permitida para cada árvore.
  - `eta`: taxa de aprendizado que controla a contribuição de cada árvore ao modelo final.
  - `gamma`: controla a complexidade das árvores através da penalização do crescimento do nó. Valores mais altos levam a uma poda mais agressiva.
  - `colsample_bytree`: Fração de colunas a serem amostradas aleatoriamente em cada iteração de construção da árvore.
  - `min_child_weight`: peso mínimo necessário para criar um novo nó na árvore.
  - `subsample`: fração de observações (ou linhas) a serem amostradas aleatoriamente em cada iteração de construção da árvore.

Ajuste de Parâmetros

- Caso você esqueça os parâmetros de um modelo no R, use a função `modelLookup()` do pacote `caret`.

```
library(caret)
modelLookup("C5.0")
```

```
  model parameter                   label forReg forClass probModel
1  C5.0    trials # Boosting Iterations  FALSE      TRUE       TRUE
2  C5.0     model            Model Type  FALSE      TRUE       TRUE
3  C5.0    winnow                Winnow  FALSE      TRUE       TRUE
```

```
modelLookup("rpart")
```

```
  model parameter                   label forReg forClass probModel
1 rpart        cp Complexity Parameter   TRUE      TRUE       TRUE
```

- Para saber os modelos e parâmetros disponíveis no pacote `caret`, acesse: https://topepo.github.io/caret/available-models.html

Ajuste de Parâmetros

- O processo de definir os valores mais adequados para os hiperparâmetros é chamado de *ajuste de parâmetros* (ou *tuning de parâmetros*).
- O ajuste costuma ser feito via busca em grade (*grid search*) e consiste em três etapas:

  1. Criar uma grade de hiperparâmetros possíveis a serem avaliados;
  2. Construir um modelo baseado em cada combinação de hiperparâmetros;
  3. Escolher aquele com melhor performance segundo alguma métrica.

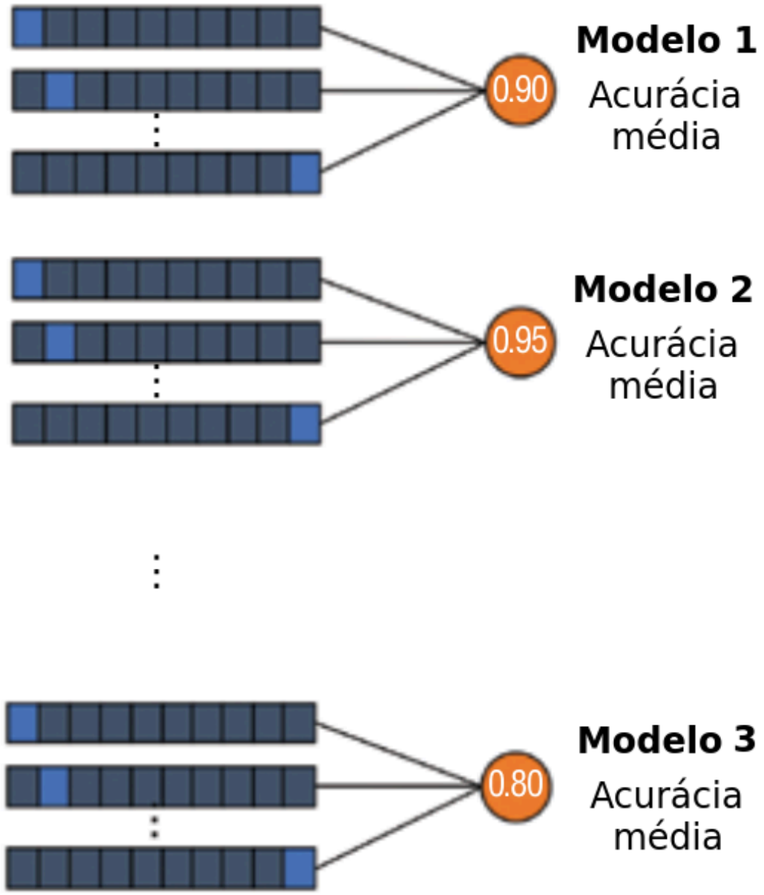
- O ajuste de parâmetros pode ser realizado de forma automática ou customizada (quando o espaço de busca é escolhido pelo usuário).

Ajuste de Parâmetros

1. Crie uma grade de parâmetros a serem considerados.

Model	Alpha	Beta	Gamma
1	1	A	TRUE
2	1	A	FALSE
3	1	B	TRUE
4	1	B	FALSE
5	2	A	TRUE
6	2	A	FALSE
7	2	B	TRUE
8	2	B	FALSE

2. Use k-fold cross validation para avaliar a performance de cada modelo.



3. Selecione o modelo com a combinação que aprensenta melhor performance.

Model	Alpha	Beta	Gamma	Accuracy
1	1	A	TRUE	0.90
2	1	A	FALSE	0.95
3	1	B	TRUE	0.92
4	1	B	FALSE	0.91
5	2	A	TRUE	0.91
6	2	A	FALSE	0.90
7	2	B	TRUE	0.89
8	2	B	FALSE	0.80

Ajuste de Parâmetros

- A função `train()` do pacote `caret` possui os seguintes argumentos:
  - `form`: especifica a variável de saída e as variáveis de entrada. Ex.: `y ~ x1 + x2 + ...`
  - `data`: especifica os dados de treino.
  - `metric`: especifica a métrica usada para avaliar o desempenho do modelo durante o treinamento. Usamos “RMSE” e “Rsquared” para regressão e “Accuracy” e “Kappa” para classificação.
  - `method`: especifica o método de aprendizado de máquina.
  - `trControl`: para ajuste customizado. Permite controlar o processo de treinamento, incluindo opções como validação cruzada, repetição do treinamento e seleção de modelo.
  - `tuneGrid`: valores dos hiperparâmetros do método a serem testados.

Ajuste de Parâmetros

Exemplo:

- Ajuste automático:

```
modelo <- train(
  y ~ .,
  data = dados_treino,
  metric = "Accuracy",
  method = "rf",
  trControl = trainControl(method = "cv", number = 10)
  )
```

Ajuste de Parâmetros

Exemplo:

- Ajuste customizado:

```
modelo <- train(
  y ~ .,
  data = dados_treino,
  metric = "Accuracy",
  method = "rf",
  tuneGrid = expand.grid(mtry = 1:10), # testar mtry de 1 a 10
  trControl = trainControl(method = "cv", number = 10)
  )
```

Agora vamos fazer no R...