

Machine Learning e Biometria Florestal: potencialidades da linguagem R

Deivison Venicio Souza

Universidade Federal Pará - UFPA
Engenheiro Florestal, Me. Ciências Florestais
Programa de Pós-graduação em Engenharia Florestal - UFPR
Especialização Data Science & Big Data (Em andamento) - UFPR
(deivisonvs@ufpa.br)



**IV Encontro Brasileiro de Mensuração Florestal
(MensuFlor)**



16 a 17/08/2018
Santa Maria, RS

Conteúdo

1 Machine Learning

2 Machine Learning no R

- Pacotes disponíveis
- Popularidade de pacotes

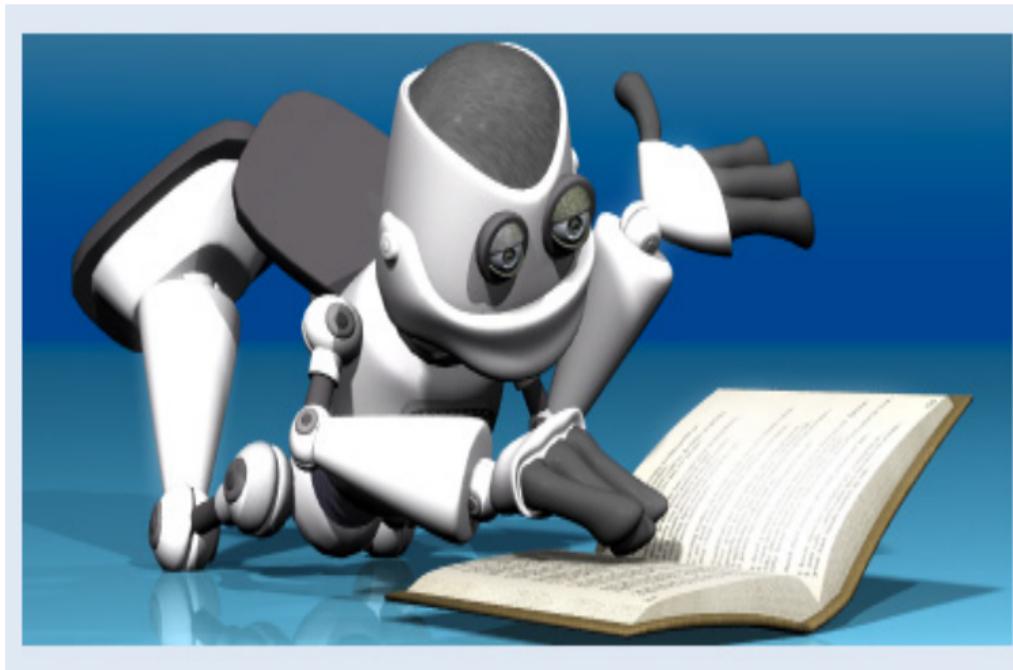
3 Pacote Caret

- Dificuldades ao implementar ML no R

4 ML supervisionado

- Pré-processamento
- Divisão dos dados
- Treinamento e Ajuste de Hiperparâmetros

O que é ML?



ML - Definição

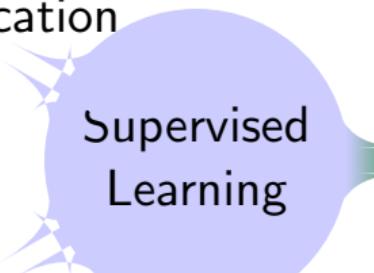
Arthur Lee Samuel: Termo "Machine Learning" em 1959.

Definição - Parafraseando Samuel (1959)

É um subconjunto de inteligência artificial que frequentemente usa técnicas estatísticas para dar aos computadores a capacidade de "aprender" com dados, sem serem explicitamente programados.

Machine Learning

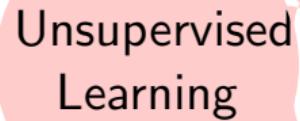
Classification



Regression

Machine
Learning

Clustering



Association

Semi-
supervised
Learning

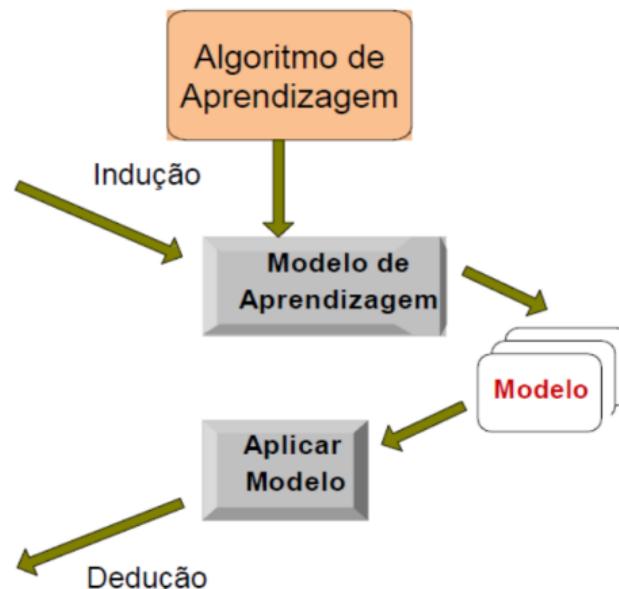
Supervised Learning

ID	Attrib1	Attrib2	Attrib3	Classe
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Conjunto de Treinamento

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Conjunto de Teste



Estudos



ELSEVIER

Contents lists available at ScienceDirect

Ecological Informatics

journal homepage: www.elsevier.com/locate/ecolinf

Automatic classification of native wood charcoal

T.M. Maruyama^a, L.S. Oliveira^a, A.S. Britto Jr^{b,c,*}, S. Nisgoski^a

Estudos

Artificial Intelligence Models to Estimate Biomass of Tropical Forest Trees

Razer Anthom Nizer Rojas Montaño, Carlos Roberto Sanquetta, Jaime Wojciechowski, Eduardo Mattar, Ana Paula Dalla Corte, and Eduardo Todt

CRAN Task View

Atualmente, são 102 pacotes sobre ML publicados no CRAN Task View.

CRAN Task View: Machine Learning & Statistical Learning
(Mantenedor: Torsten Hothorn, Versão: 21/07/2018)

ahaz	e1071	GMMBoost	LiblineaR	partykit	relaxo	spa
arules	earth	gradDescent	LogicReg	pdp	rgenoud	stabs
BART	effects	grf	LTRCtrees	penalized	RLT	SuperLearner
bartMachine	elasticnet	grplasso	maptree	penalizedLDA	Rmalschains	svmpath
BayesTree	ElemStatLearn	grpreg	mboost	picasso	rminer	tensorflow
biglasso	evclass	h2o	mlr	plotmo	rnn	tgp
bmrm	evtree	hda	model4you	quantregForest	ROCR	tree
Boruta	FCNN4R	hdi	MXM	randomForest	RoughSets	trtf
bst	frbs	hdm	ncvreg	randomForestSRC	rpart	varSelRF
C50	GAMBoost	ICEbox	nnet	ranger	RPMM	vcrpart
caret	gamboostLSS	ipred	oem	rattle	RSNNS	wsrf
CORElearn	gbm	kernlab	OneR	Rborist	RWeka	xgboost
CoxBoost	ggRandomForests	klaR	opusminer	RcppDL	RXshrink	-
Cubist	glmnet	lars	pamr	rdetools	sda	-
deepnet	glmpath	lasso2	party	REEMtree	SIS	-

Qual pacote usar?



Qual pacote usar?

Qual pacote usar?

- 1 Objetivo:** Qual(is) algoritmo(s) será(ão) testado(s)?;
 - 2 Popularidade:** Pode-se medir a popularidade dos pacotes de AM;
 - 3 Número de downloads:** Observar o número de downloads do pacote no repositório CRAN ([pacote cranlogs](#) → `cran_downloads()`); e
 - 4 Tempo de execução:** Maior velocidade de execução da tarefa → [Big Data](#).

Qual pacote usar?

- 1 Objetivo:** Qual(is) algoritmo(s) será(ão) testado(s)?;
 - 2 Popularidade:** Pode-se medir a popularidade dos pacotes de AM;
 - 3 Número de downloads:** Observar o número de downloads do pacote no repositório CRAN ([pacote cranlogs](#) → `cran_downloads()`); e
 - 4 Tempo de execução:** Maior velocidade de execução da tarefa → [Big Data](#).

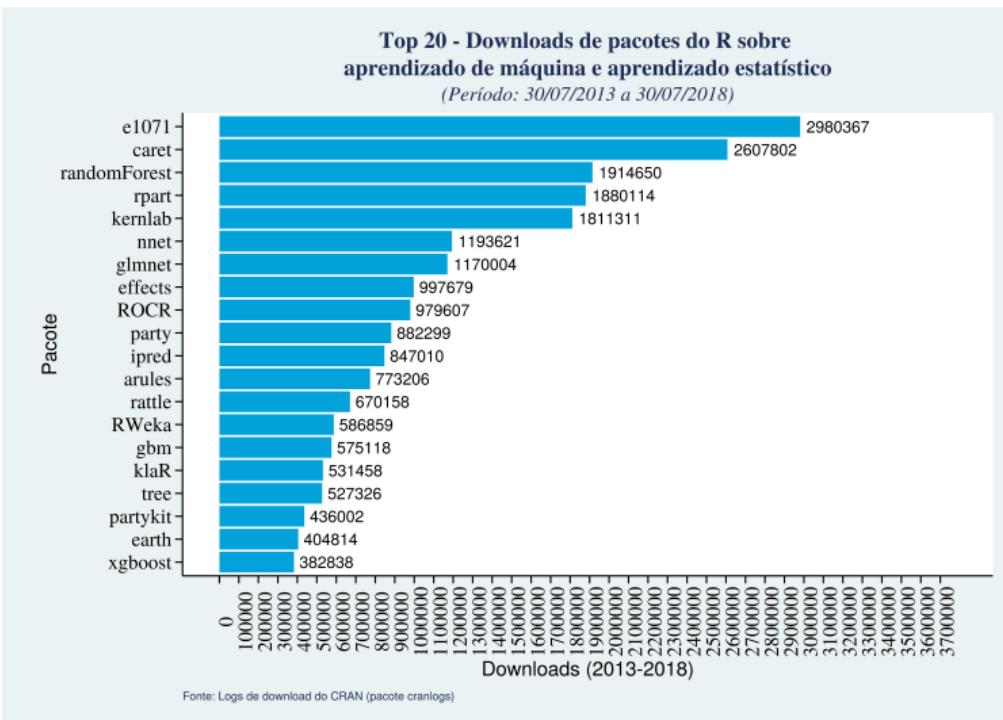
Qual pacote usar?

- 1 Objetivo:** Qual(is) algoritmo(s) será(ão) testado(s);
 - 2 Popularidade:** Pode-se medir a popularidade dos pacotes de AM;
 - 3 Número de downloads:** Observar o número de downloads do pacote no repositório CRAN ([pacote cranlogs](#) → `cran_downloads()`); e
 - 4 Tempo de execução:** Maior velocidade de execução da tarefa → [Big Data](#).

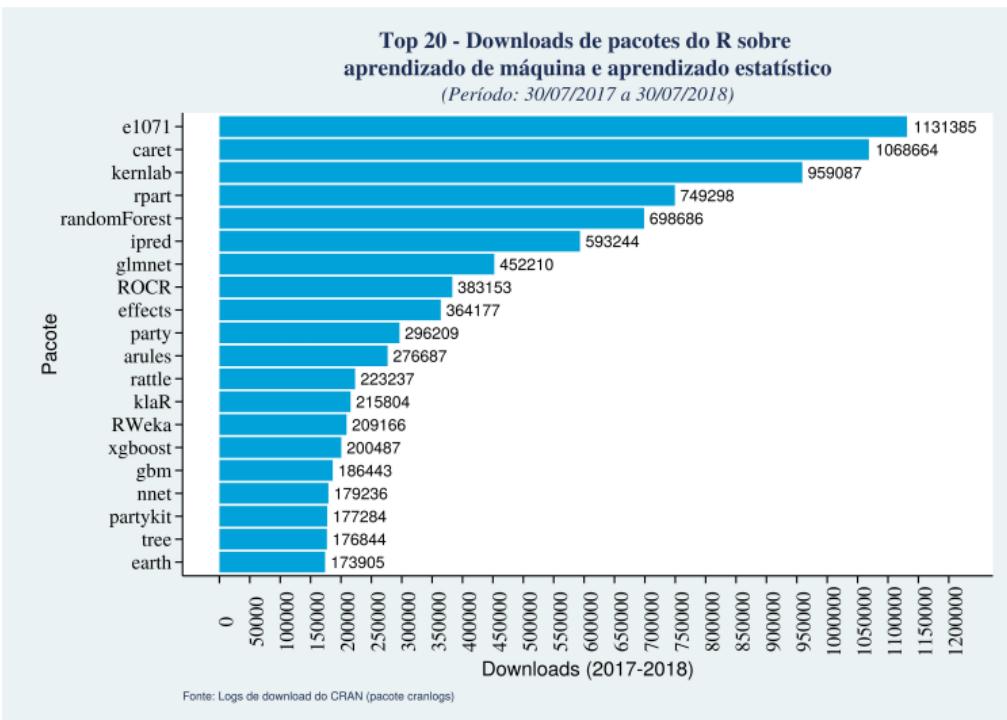
Qual pacote usar?

- 1 Objetivo:** Qual(is) algoritmo(s) será(ão) testado(s);
 - 2 Popularidade:** Pode-se medir a popularidade dos pacotes de AM;
 - 3 Número de downloads:** Observar o número de downloads do pacote no repositório CRAN ([pacote cranlogs](#) → `cran_downloads()`); e
 - 4 Tempo de execução:** Maior velocidade de execução da tarefa → [Big Data](#).

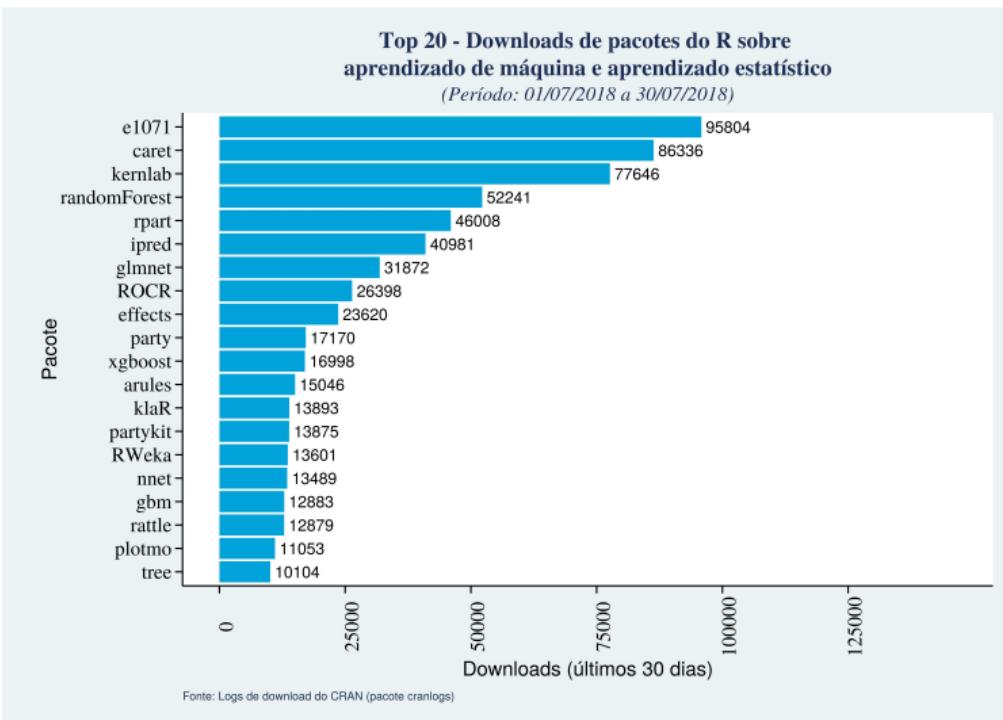
TOP 20 - Downloads (últimos 5 anos)



TOP 20 - Downloads (último ano)



TOP 20 - Downloads (30 dias)



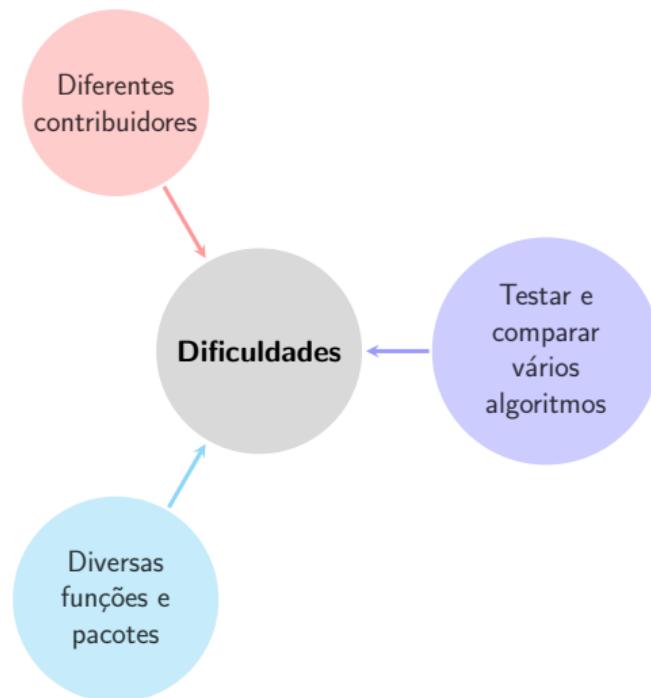
Pacote caret

(Classification And Regression Training)

"Constitui um conjunto de funções que tentam simplificar o processo de construção de modelos preditivos."

web page: <https://topepo.github.io/caret/index.html>

Pacote Caret



Pacote Caret

O pacote **caret** surge com o objetivo de: (KUHN, 2008)

- **Sintaxe:** Eliminar diferenças sintáticas entre muitas funções de construção de modelos preditivos;
- **Abordagens:** Desenvolver um conjunto de abordagens semi-automatizadas para otimizar os valores de hiperparâmetros de ajuste;
- **Processamento Paralelo:** Criar um pacote que possa ser facilmente estendido para sistemas de processamento paralelo.

Pacote Caret

O pacote **caret** surge com o objetivo de: (KUHN, 2008)

- **Sintaxe:** Eliminar diferenças sintáticas entre muitas funções de construção de modelos preditivos;
- **Abordagens:** Desenvolver um conjunto de abordagens semi-automatizadas para otimizar os valores de hiperparâmetros de ajuste;
- **Processamento Paralelo:** Criar um pacote que possa ser facilmente estendido para sistemas de processamento paralelo.

Pacote Caret

O pacote **caret** surge com o objetivo de: (KUHN, 2008)

- **Sintaxe:** Eliminar diferenças sintáticas entre muitas funções de construção de modelos preditivos;
- **Abordagens:** Desenvolver um conjunto de abordagens semi-automatizadas para otimizar os valores de hiperparâmetros de ajuste;
- **Processamento Paralelo:** Criar um pacote que possa ser facilmente estendido para sistemas de processamento paralelo.

Pacote Caret

O pacote **caret** surge com o objetivo de: (KUHN, 2008)

- **Sintaxe:** Eliminar diferenças sintáticas entre muitas funções de construção de modelos preditivos;
 - **Abordagens:** Desenvolver um conjunto de abordagens semi-automatizadas para otimizar os valores de hiperparâmetros de ajuste;
 - **Processamento Paralelo:** Criar um pacote que possa ser facilmente estendido para sistemas de processamento paralelo.

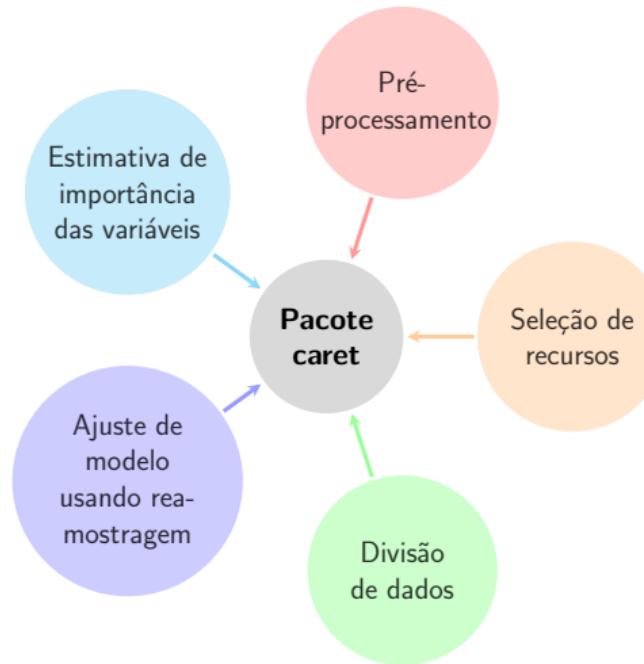
Pacote Caret

Por que usar o pacote `caret`?



Pacote Caret

Em termos gerais, o pacote **caret** possui ferramentas para:



Pacote Caret

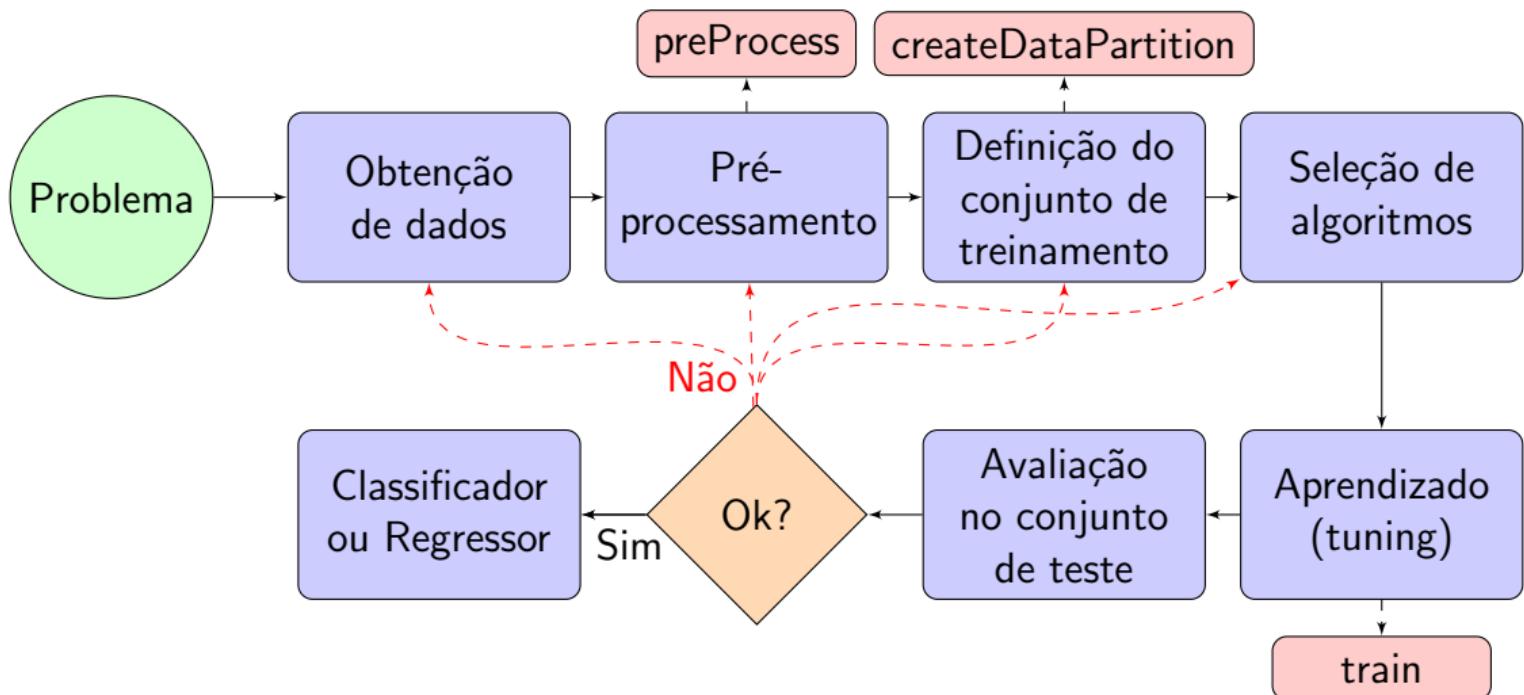
Função	Tarefa
findCorrelation()	Encontrar variáveis altamente correlacionadas
nearZeroVar()	Identificar preditores com variância próxima de zero
preProcess()	Realizar pré-processamento
createDataPartition()	Dividir aleatoriamente o conjunto de dados (estratificação)
train()	Ajustar modelos preditivos utilizando reamostragem
varImp()	Estimar a importância das variáveis preditoras
resamples()	Agrupar e visualizar os resultados da reamostragem
diff.resamples()	Fazer inferências sobre diferenças de desempenho de modelos
confusionMatrix()	Criar uma matriz de confusão
plotObsVsPred()	Gerar gráfico de valores observados versus preditos

Processo SML

(Supervised Machine Learning)

Uma abordagem usando o pacote [caret](#).
(KUHN; JOHNSON, 2013; KUHN, 2018)

Processo SML - Simplificado



Pré-processamento

Função preProcess()

Pré-processamento

Definição

As técnicas de pré-processamento de dados geralmente se referem à **adição**, **exclusão** ou **transformação** dos dados do conjunto de treinamento (KUHN; JOHNSON, 2013), previamente à construção do modelo preditivo.

Pré-processamento

Por que?

Os diferentes modelos possuem **sensibilidades distintas** para os tipos de preditores. Assim, a forma com que os preditores entram no modelo também é importante (KUHN; JOHNSON, 2013).

A transformação de preditores é estratégica para diminuir os efeitos de variáveis com **maiores escalas** de medidas sobre a determinação das **métricas de distância** (WITTEN et al., 2017).

Pré-processamento

Por que?

Os diferentes modelos possuem **sensibilidades distintas** para os tipos de preditores. Assim, a forma com que os preditores entram no modelo também é importante (KUHN; JOHNSON, 2013).

A transformação de preditores é estratégica para diminuir os efeitos de variáveis com **maiores escalas** de medidas sobre a determinação das **métricas de distância** (WITTEN et al., 2017).

Pré-processamento

Por que?

Os diferentes modelos possuem **sensibilidades distintas** para os tipos de preditores. Assim, a forma com que os preditores entram no modelo também é importante (KUHN; JOHNSON, 2013).

A transformação de preditores é estratégica para diminuir os efeitos de variáveis com **maiores escalas** de medidas sobre a determinação das **métricas de distância** (WITTEN et al., 2017).

Pré-processamento

Boa prática?

Testar uma série de transformações dos dados brutos combinadas com vários algoritmos de aprendizado de máquina, pode ajudar na descoberta de boas representações dos dados e algoritmos melhores capazes de explorar a estrutura dessas representações (BROWNLEE, 2017).

Algoritmos: k -nearest neighbors, Support Vector Machines, Artificial Neural Networks.

Pré-processamento

Função preProcess() - métodos

a) Métodos para padronização ou normalização de variáveis preditoras:

- "center": Subtrai cada valor x_i da média $\bar{x} = (x_i - \bar{x})$.
- "scale": Divide cada valor x_i pelo desvio padrão $sd_{(x)} = (x_i / sd_{(x)})$.
- "center" e "scale": Padroniza os dados ($\bar{x} = 0$ e $sd_{(x)} = 1$).
- "range": Dimensiona os dados no intervalo [0, 1].

b) Métodos para transformação → distribuição mais simétrica:

- "BoxCox": Uma transformação Box-Cox (diferentes de zero e positivos).

Pré-processamento

Função preProcess() - métodos

a) Métodos para padronização ou normalização de variáveis preditoras:

- "center": Subtrai cada valor x_i da média $\bar{x} = (x_i - \bar{x})$.
- "scale": Divide cada valor x_i pelo desvio padrão $sd_{(x)} = (x_i / sd_{(x)})$.
- "center" e "scale": Padroniza os dados ($\bar{x} = 0$ e $sd_{(x)} = 1$).
- "range": Dimensiona os dados no intervalo [0, 1].

b) Métodos para transformação → distribuição mais simétrica:

- "BoxCox": Uma transformação Box-Cox (diferentes de zero e positivos).

Pré-processamento

Função preProcess() - métodos

a) Métodos para padronização ou normalização de variáveis preditoras:

- "center": Subtrai cada valor x_i da média $\bar{x} = (x_i - \bar{x})$.
 - "scale": Divide cada valor x_i pelo desvio padrão $sd_{(x)} = (x_i / sd_{(x)})$.
 - "center" e "scale": Padroniza os dados ($\bar{x} = 0$ e $sd_{(x)} = 1$).
 - "range": Dimensiona os dados no intervalo $[0, 1]$.

b) Métodos para transformação → distribuição mais simétrica:

- "BoxCox": Uma transformação Box-Cox (diferentes de zero e positivos).

Pré-processamento

Função `preProcess()` - métodos

c) Métodos de imputação de dados:

- "knnImpute": Encontra os k vizinhos mais próximos (euclidiana) → média.
- "medianImpute": Imputação via medianas de cada preditor.
- "bagImpute": Imputação via bagging.

d) Outros métodos: "YeoJohnson", "expoTrans", "pca", "ica", "spatialSign", "corr", "zv", "nzv", and "conditionalX".

Obs.: A função `preProcess` apenas estima os parâmetros necessários para cada método. Em seguida, deve-se usar a função `predict.preProcess` para aplicar o(s) método(s) em conjuntos de dados específicos.

Pré-processamento

Função `preProcess()` - métodos

c) Métodos de imputação de dados:

- "knnImpute": Encontra os k vizinhos mais próximos (euclidiana) → média.
- "medianImpute": Imputação via medianas de cada preditor.
- "bagImpute": Imputação via bagging.

d) Outros métodos: "YeoJohnson", "expoTrans", "pca", "ica", "spatialSign", "corr", "zv", "nzv", and "conditionalX".

Obs.: A função `preProcess` apenas estima os parâmetros necessários para cada método. Em seguida, deve-se usar a função `predict.preProcess` para aplicar o(s) método(s) em conjuntos de dados específicos.

Pré-processamento

Função `preProcess()` - métodos

c) Métodos de imputação de dados:

- "knnImpute": Encontra os k vizinhos mais próximos (euclidiana) → média.
- "medianImpute": Imputação via medianas de cada preditor.
- "bagImpute": Imputação via bagging.

d) Outros métodos: "YeoJohnson", "expoTrans", "pca", "ica", "spatialSign", "corr", "zv", "nzv", and "conditionalX".

Obs.: A função `preProcess` apenas estima os parâmetros necessários para cada método. Em seguida, deve-se usar a função `predict.preProcess` para aplicar o(s) método(s) em conjuntos de dados específicos.

Pré-processamento

Função preProcess() - Exemplo intuitivo

```
df <- data.frame(  
  especie = c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe"),  
  diametro = c(NA, 27.0, 33.6, 42.6, 52.1),  
  altura = c(8.4, 8.7, 9.1, NA, 15.4),  
  cortar = c("Não", "Não", "Não", "Não", "Sim"),  
  stringsAsFactors = TRUE)  
df
```

especie	diametro	altura	cortar
Acapu	NA	8.4	Não
Araucaria	27.0	8.7	Não
Mogno	33.6	9.1	Não
Cedro	42.6	NA	Não
Ipe	52.1	15.4	Sim

Pré-processamento

Função preProcess() - "center" → "scale"

```
# Center e Scale
library(caret)
CentScl <- preProcess(x=df[,2:3], method = c("center", "scale"))
predict(CentScl, df)
```

especie	diametro	altura	cortar
Acapu	NA	-0.5977922	Não
Araucaria	-1.0830748	-0.5081234	Não
Mogno	-0.4785680	-0.3885650	Não
Cedro	0.3457596	NA	Não
Ipe	1.2158832	1.4944806	Sim

Pré-processamento

Função preProcess() - "center" → "scale" → "knnImpute"

```
#knnImpute
knnImpute <- preProcess(x=df[,2:3], method = c("knnImpute"), k=3, knnSummary = mean)
predict(knnImpute, df)
```

especie	diametro	altura	cortar
Acapu	-0.1152532	-0.5977922	Não
Araucaria	-1.0830748	-0.5081234	Não
Mogno	-0.4785680	-0.3885650	Não
Cedro	0.3457596	0.1992641	Não
Ipe	1.2158832	1.4944806	Sim

Divisão dos dados

Função `createDataPartition()`

Divisão dos dados

Em geral, em ML reporta-se a três conjuntos de dados (WITTEN et al., 2017):

- **Conjunto de aprendizado (treino):** Usado para construir os modelos preditivos (classificação ou regressão).
 - **Conjunto de validação:** Usado para otimizar os hiperparâmetros dos modelos e escolher a configuração de melhor desempenho preditivo.
 - **Conjunto de teste:** Usado para estimar o desempenho preditivo final do modelo otimizado (*tuning model*).

Divisão dos dados

Importante - Conjuntos independentes

"Cada um dos três conjuntos deve ser escolhido de forma independente"
(WITTEN et al., 2017)

- O conjunto de validação deve ser diferente do conjunto de treinamento para obter um bom desempenho no estágio de otimização ou seleção; e
- O conjunto de teste deve ser diferente de ambos para obter uma estimativa confiável da taxa de erro real.

Divisão dos dados

Função `createDataPartition()`

A função `createDataPartition` pode ser usada para criar divisões aleatórias estratégicas (*stratified random split*) de um conjunto de dados (KUHN, 2008).

```
createDataPartition(y, times = 1, p = 0.5, list = TRUE, groups = min(5,length(y)))
```

y = variável que servirá de base para o particionamento.

times = número de partições a serem geradas.

p = porcentagem de dados do conjunto de treinamento.

list = argumento lógico indicando o formato de saída dos resultados.

Divisão dos dados

Função `createDataPartition()`

A amostragem aleatória é feita dentro dos níveis de **y**:

Se $y = \text{fator}$ → a amostragem aleatória é feita considerando os níveis de fatores. Assim, busca obter representatividade de todas as classes, bem como equilibrar a quantidade amostrada dentro das classes.

Se $y = \text{numérico}$ → a amostra é dividida em subgrupos com base em percentis. Assim, a amostragem aleatória é feita dentro desses subgrupos (KUHN et al., 2017).

Divisão dos dados

Função `createDataPartition()` - Iris Flowers Dataset

```
# Conjunto de dados iris
data(iris); head(iris[c(1:2, 51:52, 101:102),])
table <- cbind(n=table(iris$Species), prop = round(prop.table(table(iris$Species))*100, 2))
addmargins(A=table, margin = seq_along(dim(table)[1]))
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
101	6.3	3.3	6.0	2.5	virginica
102	5.8	2.7	5.1	1.9	virginica

	n	prop
setosa	50	33.33
versicolor	50	33.33
virginica	50	33.33
Sum	150	99.99



Divisão dos dados

Função `createDataPartition()` - Particionando os dados

```
# Divisão dos dados: Treinamento e Teste
library(caret)
set.seed(10)
trainIndex <- createDataPartition(iris$Species, p = .8, list = FALSE, times = 1)
trainingSet <- setDT(iris)[trainIndex,]
testSet   <- setDT(iris)[-trainIndex,]
```

ID	Species	N	Prop
Treinamento	setosa	40	0.8
Treinamento	versicolor	40	0.8
Treinamento	virginica	40	0.8
Teste	setosa	10	0.2
Teste	versicolor	10	0.2
Teste	virginica	10	0.2

Divisão dos dados

Função `createDataPartition()` - Garantia das propriedades estatísticas

Conjunto de dados completo							
	n	Min	Max	Mean	Stdev	Skewness	Kurtosis
Sepal.Length	150	4.3	7.9	5.843333	0.828066	0.308641	-0.605813
Sepal.Width	150	2.0	4.4	3.057333	0.435866	0.312615	0.138705
Petal.Length	150	1.0	6.9	3.758000	1.765298	-0.269411	-1.416857
Petal.Width	150	0.1	2.5	1.199333	0.762238	-0.100917	-1.358179

Conjunto de treinamento							Conjunto de teste								
	n	Min	Max	Mean	Stdev	Skewness	Kurtosis		n	Min	Max	Mean	Stdev	Skewness	Kurtosis
Sepal.Length	120	4.4	7.9	5.824167	0.814985	0.300240	-0.596164	Sepal.Length	30	4.3	7.7	5.920000	0.888781	0.287114	-0.849711
Sepal.Width	120	2.0	4.4	3.065833	0.442965	0.358304	0.204803	Sepal.Width	30	2.2	3.8	3.023333	0.411627	0.025169	-0.652842
Petal.Length	120	1.0	6.7	3.730833	1.748074	-0.288770	-1.449284	Petal.Length	30	1.1	6.9	3.866667	1.859242	-0.208953	-1.445671
Petal.Width	120	0.1	2.5	1.187500	0.748507	-0.107040	-1.344308	Petal.Width	30	0.1	2.4	1.246667	0.826598	-0.104294	-1.534462

Treinamento e Ajuste de Hiperparâmetros

Função train()

Treinamento e Ajuste de Hiperparâmetros

Hyperparameter

Muitos algoritmos de ML possuem "parâmetros" que podem ser ajustados (tuned) para otimizar seu desempenho. Esses parâmetros são denominados **Hiperparâmetros** (WITTEN et al., 2017; PROBST et al., 2018).

Hyperparameter tuning

O termo *hyperparameter tuning* (hyperparameter optimization) pode ser definido como o processo de encontrar boas configurações de hiperparâmetros de um algoritmo para um conjunto de dados específico (PROBST et al., 2018).

Treinamento e Ajuste de Hiperparâmetros



Treinamento e Ajuste de Hiperparâmetros

Alguns métodos e seus hiperparâmetros de ajuste:

Modelo	Método	*Tarefa	Pacote	Hiperparâmetro
k-Nearest Neighbors	knn	C, R	caret	k
SVM with Linear Kernel	svmLinear	C, R	kernlab	C
SVM with Linear Kernel	svmLinear2	C, R	e1071	cost
Multi-Layer Perceptron	mlp	C, R	RSNNS	size
Neural Network	neuralnet	R	neuralnet	layer1,layer2,layer3
Neural Network	nnet	C, R	nnet	size, decay
Ridge Regression	ridge	R	elasticnet	lambda
CART	rpart	C, R	rpart	cp
Random Forest	rf	C, R	randomForest	mtry

*C = Classificação; R = Regressão

Treinamento e Ajuste de Hiperparâmetros

Função train()

A função `train()` possui as seguintes utilidades:

- **Capacidade preditiva:** Obter a estimativa da capacidade preditiva (performance) para diferentes combinações de hiperparâmetros (candidatos) com base na amostra de treinamento, usando técnicas de reamostragem; e
- **Modelo final:** Indicar o melhor modelo preditivo. Isto é, aquele com hiperparâmetros de ajuste ótimo (optimal tuning hyperparameters). Este é o modelo final (tuning model).

Treinamento e Ajuste de Hiperparâmetros

Função train()

Parâmetros:

- **x**: matriz ou dataframe que contém as variáveis preditoras;
- **y**: vector que contém a variável de resposta;
- **form**: formula para indicar as variáveis preditoras e a resposta;
- **data**: dataframe que contém o conjunto de dados;
- **method**: método de construção do modelo preditivo (algoritmos);
- **preProcess**: pré-processamento das variáveis preditoras;
- **metric**: métrica de avaliação da capacidade preditiva dos modelos ("RMSE", "Rsquared", "Precisão", "ROC" ...)

Treinamento e Ajuste de Hiperparâmetros

Função train()

Parâmetros (Cont.):

- **trControl**: controlar a construção do modelo e definir a técnica de reamostragem;
- **tuneGrid**: receber um dataframe com os candidatos a hiperparâmetro de ajuste ótimo; e
- **tuneLength**: número de níveis para cada hiperparâmetro de ajuste (usar apenas caso tuneGrid não esteja especificado).

Treinamento e Ajuste de Hiperparâmetros

Função train() - Parâmetro trControl()

```
trainControl(method="repeatedcv",number=10,repeats=10)
```

- method:

- {
 - "*none*" = ajusta um único modelo para todo conj. treino.
 - "*cv*" = k-fold cross-validation.
 - "*repeatedcv*" = repeated k-fold cross-validation.
 - "*boot*" = bootstrapping.
 - "*LOOCV*" = leave-one-out cross-validation.

Treinamento e Ajuste de Hiperparâmetros

Função train() - Esquema de treinamento

```
1 Define sets of model parameter values to evaluate
2 for each parameter set do
3   for each resampling iteration do
4     Hold-out specific samples
5     [Optional] Pre-process the data
6     Fit the model on the remainder
7     Predict the hold-out samples
8   end
9   Calculate the average performance across hold-out predictions
10 end
11 Determine the optimal parameter set
12 Fit the final model to all the training data using the optimal parameter set
```

Treinamento e Ajuste de Hiperparâmetros

Métodos de Reamostragem

- **Hold-out validation** : Separa 2/3 para treinamento e 1/3 para teste.
- **k-fold Cross-Validation:** Particiona o conj. de treinamento em k subconjuntos disjuntos ($k-1$), e testa no conjunto de validação (*hold-out*) (80%-20%).
- **Leave-one-out Cross-Validation:** $k=n$
- **Bootstrap:** Amostragem com reposição.

Navalha de Occam (Guilherme de Occam)

Dados dois modelos com erro de generalização similares, deve-se preferir o modelo mais simples em relação ao modelo mais complexo.

Considerações Finais

Desafios

Compreender melhor e avançar nas implementações de tarefas de ML nas Ciências Florestais!

Recomendação

Aprender uma linguagem de programação.



Desafios de ML

Desafios em ML

- Quantidade insuficiente de dados de treinamento;
- Conjunto de dados de treinamento não representativos;
- Dados de baixa qualidade; e
- Overfitting e Underfitting.

O limite do modelo em representar a realidade é, em boa parte, determinado pela qualidade dos dados.



Os modelos serão tão bons quanto os dados utilizados para ensinar!



Bibliografia I

- ❑ KUHN, M. Building predictive models in r using the caret package. **Journal of Statistical Software, Articles**, v. 28, n. 5, p. 1–26, 2008. ISSN 1548-7660. Disponível em: <<https://www.jstatsoft.org/v028/i05>>.
- ❑ KUHN, M. **caret: Classification and Regression Training**. [S.I.], 2018. R package version 6.0-79. Disponível em: <<https://CRAN.R-project.org/package=caret>>.
- ❑ KUHN, M.; JOHNSON, K. **Applied predictive modeling**. [S.I.]: Springer, 2013. v. 810.
- ❑ KUHN, M. et al. **caret: Classification and Regression Training**. [S.I.], 2017. R package version 6.0-78. Disponível em: <<https://CRAN.R-project.org/package=caret>>.

Bibliografia II

- PROBST, P. et al. Tunability: Importance of hyperparameters of machine learning algorithms. **arXiv preprint arXiv:1802.09596**, 2018.

WITTEN, I. H. et al. **Data Mining: Practical machine learning tools and techniques**. [S.I.]: Morgan Kaufmann, 2017.

OBRIGADO!

Deivison Venicio Souza (UFPA)
Email: deivisonvs@ufpa.br

