

caret Pacote

Folha de referência

Especificando o Modelo

Possíveis sintaxes para especificar as variáveis no modelo:

```
train(y ~ x1 + x2, data = dat, ...)
train(x = predictor_df, y = outcome_vector, ...)
train(recipe_object, data = dat, ...)
```

- `rfe`, `sbfi`, `gafs`, e `safs` possuem somente x/y interface.
- A fórmula do método `train` sempre criará variáveis *dummy*.
- A x/y interface para `train` não criará variáveis *dummy* (porém a função submetida ao modelo talvez faça).

Lembre-se:

- De nomear colunas em seus dados.
- De usar fatores para o resultado de classificação (não 0/1 ou inteiros).
- De ter nomes válidos em R para classificar níveis (não "0"/"1")
- Defina o número aleatório antes de executar `train` repetidamente antes de obter as mesmas reamostragens.
- Use o `train` com opção `na.action = na.pass` se você for importar dados ausentes. Também use a opção para predição de novos dados ausentes.

Para passar opções para a função do modelo, você pode executá-las em `train` via *ellipses*:

```
train(y ~ ., data = dat, method = "rf",
      # options to `randomForest`:
      importance = TRUE)
```

Processamento Paralelo

O pacote `foreach` é utilizado para executar o modelo em paralelo. O código de `train` não muda, porém um pacote "do" deve ser acionado primeiro.

```
# em MacOS ou Linux      # em Windows
library(doMC)             library(doParallel)
registerDoMC(cores=4)      cl <- makeCluster(2)
                           registerDoParallel(cl)
```

A função `parallel::detectCores` também pode ajudar.

Pré-processamento

Transformações, filtros e outras operações podem ser aplicadas aos *preditores* com a opção `preProc`.

```
train(preProc = c("method1", "method2"), ...)
```

Métodos incluem:

- "center", "scale" e "range" para normalizar os preditores.
- "BoxCox", "YeoJohnson" ou "expoTrans" para transformar os preditores.
- "knnImpute", "bagImpute" ou "medianImpute" para importar.
- "corr", "nzv", "zv" e "conditionalX" para filtrar.
- "pca", "ica" ou "spatialSign" para transformar grupos.

`train` determina a ordem das operações; a ordem que os métodos são declarados não importa.

O pacote `recipes` possui uma lista mais extensa de operações de pré-processamento.

Opções adicionais

Muitas opções de `train` podem ser especificadas utilizando a função `trainControl`:

```
train(y ~ ., data = dat, method = "cubist",
      trControl = trainControl(<options>))
```

Opções de reamostragem

`trainControl` é utilizado para escolher um método de reamostragem:

```
trainControl(method = <method>, <options>)
```

Métodos e opções são:

- "cv" para K-fold validação cruzada (`number` define o n° de partes).
- "repeatedcv" para validação cruzada repetida (`repeats` para n° de repetições).
- "boot" para bootstrap (`number` define as interações).
- "LGOVCV" para leave-group-out (`number` e `p` são opções).
- "LOO" para validação cruzada leave-one-out.
- "oob" para reamostragem out-of-bag (somente para alguns modelos).
- "timeslice" para dados temporais (as opções são `initialWindow`, `horizon`, `fixedWindow` e `skip`).

Métricas de performance

Para escolher como resumir o modelo, a função `trainControl` é utilizada novamente.

```
trainControl(summaryFunction = <R function>,
              classProbs = <logical>)
```

Funções configuradas do R podem ser usadas, mas o pacote `caret` inclui diversas: `defaultSummary` (para acurácia, RMSE, etc), `twoClassSummary` (para curvas ROC) e `prSummary` (para recuperação de informação). Para as duas últimas funções, a opção `classProbs` deve ser definida como `TRUE`.

Grid Search

Para permitir que `train` determine os valores de otimização, a opção `tuneLength` controla quantos valores por parâmetro **per tuning** para avaliar.

Alternativamente, valores específicos de hiperparâmetros podem ser declarados utilizando o argumento `tuneGrid`:

```
grid <- expand.grid(alpha = c(0.1, 0.5, 0.9),
                   lambda = c(0.001, 0.01))
```

```
train(x = x, y = y, method = "glmnet",
      preProc = c("center", "scale"),
      tuneGrid = grid)
```

Random Search

Para otimização, `train` pode também gerar combinações aleatórias de parâmetros de ajuste em uma ampla faixa.

`tuneLength` controla o total de combinações para avaliar.

Para usar random search:

```
trainControl(search = "random")
```

Subamostragem

Com uma grande classe desequilibrada, `train` pode subamostrar os dados para balancear as classes anterior ao ajuste do modelo.

```
trainControl(sampling = "down")
```

Outros valores são "up", "smote" ou "rose". Os dois últimos talvez necessitem instalar pacotes adicionais.