Paquete caret

Guía Rápida

Especificando el Modelo

Posibles sintaxis para especificar las variables en el modelo:

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

- `rfe`, `sbf`, `gafs`, y `safs` solo tienen el interfaz `x/y`.
- El método para la formula `train` creará **siempre** variables dummy
- El interfaz `x/y` de `train` no creará variables dummy (pero la función del modelo representado podría hacerlo).

Recuerda:

- Tener nombres en las columnas de tus datos.
- Usar factores para las variables objetivo en los modelos de clasificación (no usar 0/1 o enteros).
- Tener nombres de R válidos para los niveles de las clases (no "0"/"1")
- Fijar una semilla aleatoria antes de llamar a la función `train` repetidamente para obtener las mismas muestras en las diferentes llamadas.
- Usar la opción `na.action = na.pass` de `train` si vas a imputar valores faltantes. También, usa esta opción cuando vayas a predecir nuevos datos que contengan faltantes.

Para pasar opciones a la función del modelo, se puede hacer a través de `train` via *las elipses*:

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

Procesamiento en Paralelo

Se usa el paquete `foreach` para ejecutar modelos en paralelo. El código de `train` no cambia pero se tiene que llamar al "`do`" primero.

```
# en MacOS o Linux
library(doMC)
registerDoMC(cores=4)
```

```
# en Windows
library(doParallel)
cl <- makeCluster(2)
registerDoParallel(cl)
```

La función `parallel::detectCores` puede ayudar también.

Preprocesado

Se pueden aplicar transformaciones, filtros y otras operaciones a las variables *predictoras* con la opción `preProc`.

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

Incluye los siguientes métodos:

- `"center"`, `"scale"`, y `"range"` para normalizar.
- `"BoxCox"`, `"YeoJohnson"`, o `"expoTrans"` para transformar.
- `"knnImpute"`, `"bagImpute"`, o `"medianImpute"` para imputar.
- `"corr"`, `"nzv"`, `"zv"`, y `"conditionalX"` para filtrar.
- `"pca"`, `"ica"`, o `"spatialSign"` para transformar grupos.

`train` determina el orden de las operaciones: el orden en el que se declaran los métodos no importa.

El paquete `recipes` contiene una lista más extensa de operaciones de preprocesamiento.

Añadiendo Opciones

Muchas opciones de `train` se pueden especificar usando la función `trainControl`:

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

Opciones de Remuestreo

`trainControl` se usa para elegir el método de remuestreo:

```
trainControl(method = <método>, <opciones>)
```

Los métodos y las opciones son:

- `"cv"` para K-iteraciones validación-cruzada (`number` define el # de iteraciones).
- `"repeatedcv"` para validación-cruzada repetida (`repeats` para el # de repeticiones).
- `"boot"` para bootstrap (`number` define las iteraciones).
- `"LGOCV"` para dejar-grupo-fuera (`number` y `p` son opciones).
- `"LOO"` para dejar-uno-fuera validación-cruzada.
- `"oob"` para el remuestreo fuera-de-la-bolsa (solo para algunos modelos).
- `"timeslice"` para series temporales (las opciones son `initialWindow`, `horizon`, `fixedWindow`, y `skip`).

Métricas de Rendimiento

Para elegir cómo resumir un modelo, se usa de nuevo la función `trainControl`

```
trainControl(summaryFunction = <función R>,
             classProbs = <lógico>)
```

Se pueden usar funciones de R a la medida, aunque `caret` incluye varias: `defaultSummary` (para Precisión, RMSE, etc), `twoClassSummary` (para curvas ROC), y `prSummary` (para recuperación de información). Para estas dos últimas funciones, se tiene que ajustar a TRUE la opción `classProbs`.

Búsqueda en Grid

Para conseguir que `train` determine los valores de el/los parámetro/s de ajuste, la opción `tuneLength` controla cuántos valores se evalúan **por parámetro de ajuste**.

De forma alternativa, se pueden declarar valores específicos de los parámetros de ajuste usando el argumento de `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)
```

Búsqueda Aleatoria

Para ajustar `train` también se pueden generar combinaciones de parámetros de ajuste de forma aleatoria sobre un rango. `tuneLength` controla el número total de combinaciones a evaluar. Para usar la búsqueda aleatoria:

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

Submuestreos

Para un gran desbalanceo entre clases, `train` puede submuestrear los datos para balancear las clases antes de ajustar el modelo.

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

Otros valores son `"up"`, `"smote"`, o `"rose"`. Los dos últimos pueden requerir la instalación de paquetes adicionales.