Le package caret

Aide-Mémoire

Spécifier Le Modèle

Syntaxes possibles pour spécifer les variables dans le modèle:

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

- rfe, sbf, gafs, et safs ont juste l'interface x/y.
- La méthode formule de train créera toujours des variables muettes.
- L'interface x/y de train ne créera pas de variables muettes (mais la fonction de modèle peut le faire).

N'oubliez pas de:

- Avoir les noms de colonnes dans votre jeu de données.
- Utiliser les factors pour la variable à classifier (pas 0/1 ou des entiers).
- Avoir des noms R valides pour les modalités (pas “0"/"1")
- Spécifier la graine aléatoire avant d'utiliser train de manière répetée afin d'avoir les mêmes échantillons à travers differents appels.
- Utiliser l'option de train na.action = na.pass si vous imputez des valeurs manquantes. Cette option est aussi utilisée pour prédire de nouvelles données avec valeurs manquantes.

Pour passer des options à la fonction du modèle sous-jacent, vous pouvez les spécifier dans la fonction train grâce aux *ellipses*:

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

Traitement Parallèle

Le package foreach permet de faire tourner des modèles en parallèle. Le code de train ne change pas mais un des packages “do” doit être chargé au préalable.

```
# sur MacOS ou Linux
library(doMC)
registerDoMC(cores=4)
```

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

La fonction parallel::detectCores peut aider aussi.

Prétraitement

Transformations, filtres, et autres operations peuvent être appliqués aux *predicteurs* avec l'option preProc.

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

Les méthodes suivantes :

- "center", "scale", et "range" pour normaliser les predicteurs.
- "BoxCox", "YeoJohnson", ou "expoTrans" pour transformer les predicteurs.
- "knnImpute", "bagImpute", ou "medianImpute" pour imputer.
- "corr", "nzv", "zv", et "conditionalX" pour filtrer.
- "pca", "ica", or "spatialSign" pour transformer des groupes.

train détermine l'ordre des opérations; l'ordre dans lequel les méthodes sont declarées ne compte pas.

Le package recipes contient une liste plus exhaustive d'operations de prétraitement.

Rajouter des Options

Beaucoup d'options de train peuvent être specifiées en utilisant la fonction trainControl:

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

Options de ré-échantillonage

trainControl est utilisé pour choisir une méthode de ré-échantillonnage:

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

Les méthodes et options sont:

- "cv" pour la “K-fold” validation croisée (number donne le # d'échantillons).
- "repeatedcv" pour la validation croisée répétée (repeats pour le # répétitions).
- "boot" pour le bootstrap (number donne le # d'iterations).
- "LGOCV" pour le “leave-group-out” (number et p sont les options).
- "LOO" pour la validation croisee “leave-one-out”
- "oob" pour le ré-échantillonnage “out-of-bag”.
- "timeslice" pour les series temporelles (les options sont initialWindow, horizon, fixedWindow, et skip).

Métrique de Performance

Pour résumer le modèle, la fonction trainControl est encore utilisée.

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

Des fonctions R personnalisées peuvent être utilisés mais caret en inclus plusieurs: defaultSummary (pour accuracy, RMSE, etc), twoClassSummary (pour les courbes ROC), et prSummary (pour récupérer des informations). Pour les deux dernieres fonctions, l'option classProbs doit être réglé à TRUE.

Recherche dans une grille

Pour que train détermine la valeur des paramètre(s) de complexité, l'option tuneLength contrôle combien de valeurs par paramètre de complexité à evaluer.

Alternativement, des valeurs spécifiques des paramètres de compléxités peuvent être déclarées en utilisant l'argument 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)
```

Recherche Aléatoire

Pour l'optimisation des hyperparamètres, train peut générer des combinaisons de paramètres aléatoires sur une large gamme. tuneLength contrôle le nombre total de combinaisons à évaluer. Pour utiliser la recherche aléatoire:

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

Sous-échantillonage

En présense de classes très désequilibrées, train peut sous-échantilloner les données afin d'équilibrer les classes avant de passer à la modélisation.

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

Les autres valeurs sont "up", "smote", ou "rose". La dernière valeur requiert l'installation de packages additionnels.