12 Using Recipes with train

Modeling functions in R let you specific a model using a formula, the `x` / `y` interface, or both. Formulas are good because they will handle a lot of minutia for you (e.g. dummy variables, interactions, etc) so you don't have to get your hands dirty. They work pretty well but also have limitations too. Their biggest issue is that not all modeling functions have a formula interface (although `train` helps solve that).

Recipes are a third method for specifying model terms but also allow for a broad set of preprocessing options for encoding, manipulating, and transforming data. They cover a lot of techniques that formulas cannot do naturally.

Recipes can be built incrementally in a way similar to how `dplyr` or `ggplot2` are created. The package website has examples of how to use the package and lists the possible techniques (called *steps*). A recipe can then be handed to `train` *in lieu* of a formula.

12.1 Why Should you learn this?

Here are two reasons:

12.1.1 More versatile tools for preprocessing data

`caret`’s preprocessing tools have a lot of options but the list is not exhaustive and they will only be called in a specific order. If you would like

- a broader set of options,
- the ability to write your own preprocessing tools, or
- to call them in the order that you desire

then you can use a recipe to do that.

12.1.2 Using additional data to measure performance

In most modeling functions, including `train`, most variables are consigned to be either predictors or outcomes. For recipes, there are more options. For example, you might want to have specific columns of your data set be available when you compute how well the model is performing, such as:

- if different stratification variables (e.g. patients, ZIP codes, etc) are required to do correct summaries or
- ancillary data might be need to compute the expected profit or loss based on the model results.

To get these data properly, they need to be made available and handled the same way as all of the other data. This means they should be sub- or resampled as all of the other data. Recipes let you do that.

12.2 An Example

The `QSARdata` package contains several chemistry data sets. These data sets have rows for different potential drugs (called "compounds" here). For each compound, some important characteristic is measured. This illustration will use the `AquaticTox` data. The outcome is called "Activity" is a measure of how harmful the compound might be to people. We want to predict this during the drug discovery phase in R&D To do this, a set of *molecular descriptors* are computed based on the compounds formula. There are a lot of different types of these and we will use the 2-dimensional MOE descriptor set. First, lets' load the package and get the data together:

```
library(caret)
library(recipes)
library(dplyr)
library(QSARdata)

data(AquaticTox)
tox <- AquaticTox_moe2D
ncol(tox)
```

```
## [1] 221
```

```
## Add the outcome variable to the data frame
tox$Activity <- AquaticTox_Outcome$Activity
```

We will build a model on these data to predict the activity. Some notes:

- A common aspect to chemical descriptors is that they are *highly correlated*. Many descriptors often measure some variation of the same thing. For example, in these data, there are 56 potential predictors that measure different flavors of surface area. It might be a good idea to reduce the dimensionality of these data by pre-filtering the predictors and/or using a dimension reduction technique.
- Other descriptors are counts of certain types of aspects of the molecule. For example, one predictor is the number of Bromine atoms. The vast majority of compounds lack Bromine and this leads to a near-zero variance situation discussed previously. It might be a good idea to pre-filter these.

Also, to demonstrate the utility of recipes, suppose that we could score potential drugs on the basis of how manufacturable they might be. We might want to build a model on the entire data set but only evaluate it on compounds that could be reasonably manufactured. For illustration, we'll assume that, as a compounds molecule weight increases, its manufacturability *decreases*. For this purpose, we create a new variable (`manufacturability`) that is neither an outcome or predictor but will be needed to compute performance.

管道函数
tox<- mutate(select(tox, -Molecule), manufactureability=(1/moe2D_Weight) /sum(1/moe2D_Weight))

```
tox <- tox %>%
  select(-Molecule) %>%
  ## Suppose the easy of manufacturability is
  ## related to the molecular weight of the compound
  mutate(manufacturability  = 1/moe2D_Weight) %>%
  mutate(manufacturability = manufacturability/sum(manufacturabil
```

For this analysis, we will compute the RMSE using weights based on the manufacturability column such that a difficult compound has less impact on the RMSE.

mutate
Mutate a data frame by adding new or replacing existing columns

```
model_stats <- function(data, lev = NULL, model = NULL) {

  stats <- defaultSummary(data, lev = lev, model = model)

  wt_rmse <- function (pred, obs, wts, na.rm = TRUE)
  sqrt(weighted.mean((pred - obs)^2, wts, na.rm = na.rm))

  res <- wt_rmse(pred = data$pred,
                 obs = data$obs,
                 wts = data$manufacturability)
  c(wRMSE = res, stats)
}
```

wt_rmse是个function，输入data $pred, data$obs, data $manufacturability分别作为pred, obs, wts用function求出赋值给res

There is no way to include this extra variable using the default `train` method or using `train.formula` .

Now, let's create a recipe incrementally. First, we will use the formula methods to declare the outcome and predictors but change the analysis role of the `manufacturability` variable so that it will only be available when summarizing the model fit.

```
tox_recipe <- recipe(Activity ~ ., data = tox) %>%
  add_role(manufacturability, new_role = "performance var")
```

```
## Warning: Changing role(s) for manufacturability
```

A recipe is a description of what steps should be applied to a data set in order to get it ready for data analysis.
recipe(x, formula = NULL, ..., vars = NULL,
  roles = NULL)

add_role can add a role definition to an existing variable in the recipe.
add_role(recipe, ..., new_role = "predictor")

```
tox_recipe

## Data Recipe
##
## Inputs:
##
##                role #variables
##             outcome          1
##     performance var          1
##           predictor        220
```

Using this new role, the `manufacturability` column will be available when the summary function is executed and the appropriate rows of the data set will be exposed during resampling. For example, if one were to debug the `model_stats` function during execution of a model, the `data` object might look like this:

```
Browse[1]> head(data)
    obs manufacturability rowIndex     pred
1  3.40       0.002770707        3 3.376488
2  3.75       0.002621364       27 3.945456
3  3.57       0.002697900       33 3.389999
4  3.84       0.002919528       39 4.023662
5  4.41       0.002561416       53 4.482736
6  3.98       0.002838804       54 3.965465
```

More than one variable can have this role so that multiple columns can be made available.

Now let’s add some steps to the recipe First, we remove sparse and unbalanced predictors:

```
tox_recipe <- tox_recipe %>% step_nzv(all_predictors())
tox_recipe
```

```
## Data Recipe
##
## Inputs:
##
##              role #variables
##           outcome          1
##   performance var          1
##         predictor        220
##
## Operations:
##
## Sparse, unbalanced variable filter on all_predictors()
```

Note that we have only specified what *will happen once the recipe* is executed. This is only a specification that uses a generic declaration of `all_predictors`.

As mentioned above, there are a lot of different surface area predictors and they tend to have very high correlations with one another. We'll add one or more predictors to the model in place of these predictors using principal component analysis. The step will retain the number of components required to capture 95% of the information contained in these 56 predictors. We'll name these new predictors `surf_area_1` , `surf_area_2` etc.

```
tox_recipe <- tox_recipe %>%
  step_pca(contains("VSA"), prefix = "surf_area_",  threshold = .
```

Now, lets specific that the third step in the recipe is to reduce the number of predictors so that no pair has an absolute correlation greater than 0.90. However, we might want to keep the surface area principal components so we *exclude* these from the filter (using the minus sign)

```
tox_recipe <- tox_recipe %>%
  step_corr(all_predictors(), -starts_with("surf_area_"), thresho
```

Finally, we can center and scale all of the predictors that are available at the end of the recipe:

```r
tox_recipe <- tox_recipe %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
tox_recipe
```

5种方法对数据预处理
step_nzv()
step_pca()
step_corr()
step_center()
step_scale()

```
## Data Recipe
##
## Inputs:
##
##             role #variables
##          outcome          1
##  performance var          1
##        predictor        220
##
## Operations:
##
## Sparse, unbalanced variable filter on all_predictors()
## PCA extraction with contains("VSA")
## Correlation filter on 2 items
## Centering for all_predictors()
## Scaling for all_predictors()
```

Let's use this recipe to fit a SVM model and pick the tuning parameters that minimize the weighted RMSE value:

```
tox_ctrl <- trainControl(method = "cv", summaryFunction = model_s
set.seed(888)
tox_svm <- train(tox_recipe, tox,
                 method = "svmRadial",
                 metric = "wRMSE",
                 maximize = FALSE,
                 tuneLength = 10,
                 trControl = tox_ctrl)
tox_svm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 322 samples
## 221 predictors
##
## Recipe steps: nzv, pca, corr, center, scale
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 290, 290, 290, 290, 290, 289, ...
## Resampling results across tuning parameters:
##
##   C       wRMSE      RMSE       Rsquared   MAE
##     0.25  0.7957625  0.7546812  0.6903770  0.5230607
##     0.50  0.7429397  0.7020489  0.7231226  0.4796514
##     1.00  0.7036632  0.6624582  0.7492423  0.4450277
##     2.00  0.6826542  0.6384319  0.7653452  0.4174271
##     4.00  0.6645231  0.6256636  0.7712178  0.4109508
##     8.00  0.6343363  0.6066267  0.7813932  0.4057199
##    16.00  0.6207548  0.6018964  0.7819211  0.4107397
##    32.00  0.6283098  0.6098797  0.7755607  0.4177769
##    64.00  0.6391904  0.6208031  0.7672873  0.4294652
##   128.00  0.6620628  0.6384815  0.7529142  0.4437434
##
## Tuning parameter 'sigma' was held constant at a value of 0.012
## wRMSE was used to select the optimal model using the smallest
## The final values used for the model were sigma = 0.01221576 an
```

What variables were generated by the recipe?

```
## originally:
ncol(tox) - 2
```

```
## [1] 220
```

```
## after the recipe was executed:
predictors(tox_svm)
```

```
##  [1] "moeGao_Abra_R"        "moeGao_Abra_acidity"  "moeGao_Abr
##  [4] "moeGao_Abra_pi"       "moe2D_BCUT_PEOE_3"    "moe2D_BCUT_
##  [7] "moe2D_BCUT_SLOGP_1"   "moe2D_BCUT_SLOGP_3"   "moe2D_GCUT_
## [10] "moe2D_GCUT_PEOE_1"    "moe2D_GCUT_PEOE_2"    "moe2D_GCUT_
## [13] "moe2D_GCUT_SLOGP_1"   "moe2D_GCUT_SLOGP_2"   "moe2D_GCUT_
## [16] "moe2D_GCUT_SMR_0"     "moe2D_Kier3"          "moe2D_Kier
## [19] "moe2D_KierA2"         "moe2D_KierA3"         "moe2D_Kier
## [22] "moe2D_PEOE_PC..1"     "moe2D_PEOE_RPC."      "moe2D_PEOE_
## [25] "moe2D_Q_PC."          "moe2D_Q_RPC."         "moe2D_Q_RP
## [28] "moe2D_SlogP"          "moe2D_TPSA"           "moe2D_Weig
## [31] "moe2D_a_ICM"          "moe2D_a_acc"          "moe2D_a_hy
## [34] "moe2D_a_nH"           "moe2D_a_nN"           "moe2D_a_nO
## [37] "moe2D_b_1rotN"        "moe2D_b_1rotR"        "moe2D_b_do
## [40] "moe2D_balabanJ"       "moe2D_chi0v"          "moeGao_chi
## [43] "moeGao_chi3cv_C"      "moeGao_chi3pv"        "moeGao_chi
## [46] "moeGao_chi4cav_C"     "moeGao_chi4pc"        "moeGao_chi
## [49] "moeGao_chi4pcv_C"     "moe2D_density"        "moe2D_kS_a
## [52] "moe2D_kS_aaaC"        "moe2D_kS_aasC"        "moe2D_kS_d
## [55] "moe2D_kS_dsCH"        "moe2D_kS_dssC"        "moe2D_kS_s
## [58] "moe2D_kS_sCl"         "moe2D_kS_sNH2"        "moe2D_kS_s
## [61] "moe2D_kS_ssCH2"       "moe2D_kS_ssO"         "moe2D_kS_s
## [64] "moe2D_lip_don"        "moe2D_petitjean"      "moe2D_radi
## [67] "moe2D_reactive"       "moe2D_rings"          "moe2D_wein
## [70] "moe2D_weinerPol"      "surf_area_1"          "surf_area_
## [73] "surf_area_3"          "surf_area_4"
```

The trained recipe is available in the `train` object and now shows specific variables involved in each step:

```
tox_svm$recipe
```

```
## Data Recipe
## 
## Inputs:
## 
##              role #variables
##           outcome          1
##   performance var          1
##         predictor        220
## 
## Training data contained 322 data points and no missing data.
## 
## Operations:
## 
## Sparse, unbalanced variable filter removed moe2D_PEOE_VSA.3, .
## PCA extraction with moe2D_PEOE_VSA.0, ... [trained]
## Correlation filter removed moe2D_BCUT_SMR_0, ... [trained]
## Centering for moeGao_Abra_R, ... [trained]
## Scaling for moeGao_Abra_R, ... [trained]
```

12.3 Case Weights

For models that accept them, case weights can be passed to the model fitting routines using a role of `"case weight"` .