IST438-W5-Applications

3/27/23

Decision trees

In this application, we will train decision trees for regression and classification tasks.

Packages

We need to install `{tidymodels}` package to train decision tree models. It is one of the most famous ML package in R because it consists many tools which are used in ML process:

- `{rsample}` is used to split dataset: `initial_split()`
- `{recipes}` for feature engineering
- `{parnship}` model fitting
- `{tune}` model tuning
- `{yardstick}` model evaluation

Please use the two-step codes below: (1) install, (2) load the package.

```
#install.packages("tidymodels") # training models
#install.packages("DALEX").     # datasets
#install.packages("rpart.plot") # visualizing decision tree
library(tidymodels)
library(DALEX)
library(rpart.plot)
```

Dataset

`apartments` and `titanic` datasets are used in application to compare the performance of the regression models and decision trees.

Model training with {tidymodels} for regression task

Data splitting:

```
apartments_split <- initial_split(data = apartments, # dataset to split
                                  prop = 0.80)    # proportion of train set

apartments_train <- apartments_split |> training()
apartments_test  <- apartments_split |> testing()
```

Model specification:

- `type`: model type, e.g. regression, decision tree or etc.
- `engine`: different R packages have engines
- `mode`: learning task, e.g. regression or classification

Defining model specification:

```
dt_model <- decision_tree() |> # try linear_reg()
  set_engine("rpart") |>       # and lm
  set_mode("regression")
```

Model training:

```
dt_apartments <- dt_model |>
  fit(m2.price ~., data = apartments_train)
dt_apartments
```

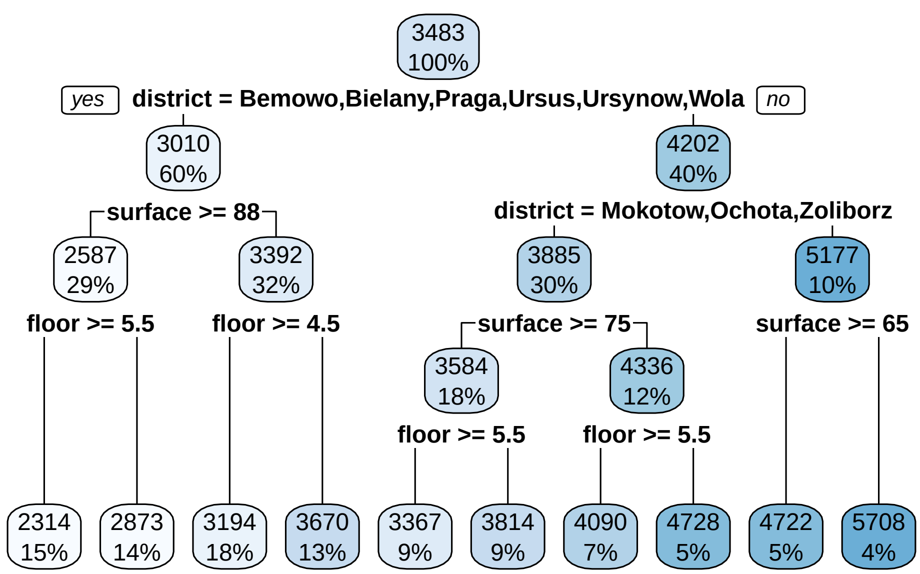
```
parsnip model object

n= 800

node), split, n, deviance, yval
      * denotes terminal node

 1) root 800 658070200 3483.499
   2) district=Bemowo,Bielany,Praga,Ursus,Ursynow,Wola 482 172944900 3009.589
     4) surface>=87.5 229  49155870 2587.393
       8) floor>=5.5 117  15143270 2313.538 *
       9) floor< 5.5 112  16071740 2873.473 *
     5) surface< 87.5 253  46022840 3391.735
```

```
      10) floor>=4.5 148  17768190 3194.399 *
      11) floor< 4.5 105  14367660 3669.886 *
   3) district=Mokotow,Ochota,Srodmiescie,Zoliborz 318 212792000 4201.814
     6) district=Mokotow,Ochota,Zoliborz 240  81098870 3884.808
      12) surface>=74.5 144  28089320 3584.062
        24) floor>=5.5 74  12182060 3366.878 *
        25) floor< 5.5 70   8726800 3813.657 *
      13) surface< 74.5 96  20448250 4335.927
        26) floor>=5.5 59   6452186 4090.000 *
        27) floor< 5.5 37   4737697 4728.081 *
     7) district=Srodmiescie 78  33364750 5177.218
      14) surface>=64.5 42   8690634 4722.476 *
      15) surface< 64.5 36   5856217 5707.750 *
```

Visualizing the decision tree:

```
rpart.plot(dt_apartments$fit)
```

```
Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and i
To silence this warning:
    Call rpart.plot with roundint=FALSE,
    or rebuild the rpart model with model=TRUE.
```



Make predictions by using the trained model:

```r
apartments_predictions <- dt_apartments |>
  predict(new_data = apartments_test)

apartments_predictions
```

```
# A tibble: 200 x 1
   .pred
   <dbl>
 1 3367.
 2 2314.
 3 4722.
 4 4728.
 5 3194.
 6 4728.
 7 3194.
 8 3194.
 9 2314.
10 4090
# ... with 190 more rows
```

Evaluating model performance:

`{yardstick}` package is used to evaluate/measure the model performance. Its functions require a data.frame or tibble with model results. To combine the model prediction and actual/observed values of target variable in test data, `cbind()` function can be used as follows:

```r
apartments_results <- tibble(predicted = apartments_predictions$.pred,
                             actual    = apartments_test$m2.price)

apartments_results
```

```
# A tibble: 200 x 2
   predicted actual
       <dbl>  <dbl>
 1     3367.   3517
 2     2314.   2346
 3     4722.   4745
 4     4728.   3961
 5     3194.   2797
 6     4728.   5116
```

```
 7     3194.   3172
 8     3194.   3378
 9     2314.   3372
10     4090    3868
# ... with 190 more rows
```

Then we can calculate the RMSE of the model by using `rmse()` function with obligatory arguments: `truth` must be assigned with actual values, `estimate` must be assigned with predicted values of target variable.

```
apartments_results |> rmse(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        422.
```

R^2 metric can be calculated in similar manner by `rsq()` function:

```
apartments_results |> rsq(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.784
```

Streaming model fitting by `last_fit()` function. It takes a model specification, model formula, and data split object.

```
apartments_last_fit <- dt_model |>
  last_fit(m2.price ~., split = apartments_split)

apartments_last_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [800/200]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

Collecting metrics:

```r
apartments_last_fit |> collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard     422.    Preprocessor1_Model1
2 rsq     standard       0.784 Preprocessor1_Model1
```

Collecting predictions:

```r
apartments_last_fit |> collect_predictions()
```

```
# A tibble: 200 x 5
   id               .pred  .row m2.price .config
   <chr>            <dbl> <int>    <dbl> <chr>
 1 train/test split 3367.     4     3517 Preprocessor1_Model1
 2 train/test split 2314.     8     2346 Preprocessor1_Model1
 3 train/test split 4722.     9     4745 Preprocessor1_Model1
 4 train/test split 4728.    11     3961 Preprocessor1_Model1
 5 train/test split 3194.    12     2797 Preprocessor1_Model1
 6 train/test split 4728.    13     5116 Preprocessor1_Model1
 7 train/test split 3194.    17     3172 Preprocessor1_Model1
 8 train/test split 3194.    18     3378 Preprocessor1_Model1
 9 train/test split 2314.    30     3372 Preprocessor1_Model1
10 train/test split 4090     31     3868 Preprocessor1_Model1
# ... with 190 more rows
```

Model training with `{tidymodels}` for classification task

```r
set.seed(123)
titanic_split <- initial_split(data = titanic, # dataset to split
                               prop = 0.80)    # proportion of train set

titanic_train <- titanic_split |> training()
titanic_test  <- titanic_split |> testing()
```

Model specification:

- `type`: model type, e.g. regression, decision tree or etc.
- `engine`: different R packages have engines
- `mode`: learning task, e.g. regression or classification

Defining model specification:

```
dt_model <- decision_tree() |>
  set_engine("rpart") |>
  set_mode("classification")
```

Model training:

```
dt_titanic <- dt_model |>
  fit(survived ~., data = titanic_train)

dt_titanic
```

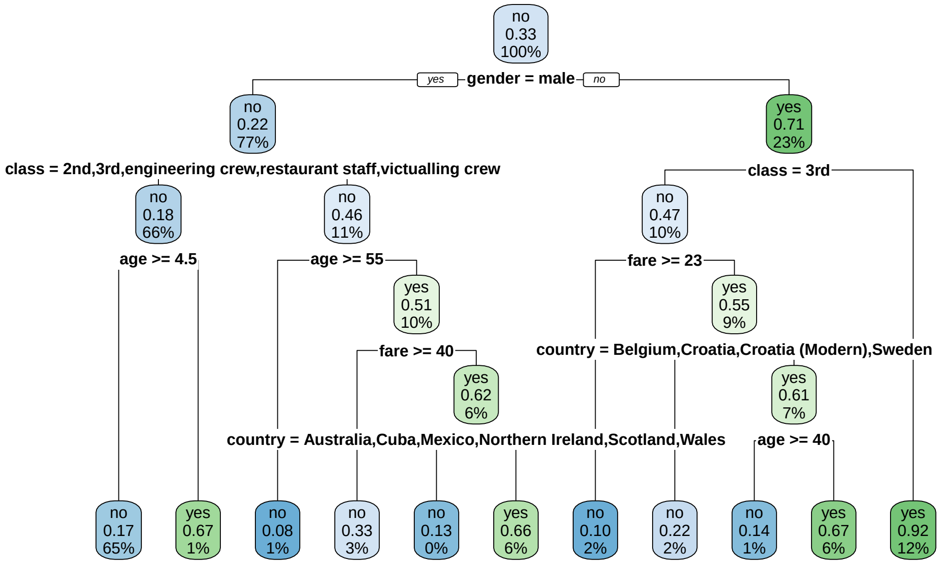
```
parsnip model object

n= 1765

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 1765 590 no (0.66572238 0.33427762)
   2) gender=male 1362 302 no (0.77826725 0.22173275)
     4) class=2nd,3rd,engineering crew,restaurant staff,victualling crew 1165 212 no (0.8180
       8) age>=4.5 1141 196 no (0.82822086 0.17177914) *
       9) age< 4.5 24   8 yes (0.33333333 0.66666667) *
     5) class=1st,deck crew 197  90 no (0.54314721 0.45685279)
      10) age>=54.5 26   2 no (0.92307692 0.07692308) *
      11) age< 54.5 171  83 yes (0.48538012 0.51461988)
        22) fare>=39.5703 61  20 no (0.67213115 0.32786885) *
        23) fare< 39.5703 110  42 yes (0.38181818 0.61818182)
          46) country=Australia,Cuba,Mexico,Northern Ireland,Scotland,Wales 8   1 no (0.8750
          47) country=Belgium,Canada,Channel Islands,England,France,Germany,Ireland,Sweden,Sw
   3) gender=female 403 115 yes (0.28535980 0.71464020)
     6) class=3rd 184  87 no (0.52717391 0.47282609)
      12) fare>=23.07 30   3 no (0.90000000 0.10000000) *
      13) fare< 23.07 154  70 yes (0.45454545 0.54545455)
        26) country=Belgium,Croatia,Croatia (Modern),Sweden 27   6 no (0.77777778 0.2222222
        27) country=Denmark,England,Finland,Ireland,Lebanon,Norway,Slovenia,Switzerland,Syri
```

```
        54) age>=39.5 14   2 no (0.85714286 0.14285714) *
        55) age< 39.5 113  37 yes (0.32743363 0.67256637) *
    7) class=1st,2nd,restaurant staff,victualling crew 219  18 yes (0.08219178 0.91780822) *
```

Visualizing the decision tree:

```r
rpart.plot(dt_titanic$fit)
```

```
Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and i
To silence this warning:
    Call rpart.plot with roundint=FALSE,
    or rebuild the rpart model with model=TRUE.
```



Make predictions by using the trained model. If you want to calculate the predicted probabilities, it is necessary to assign the `type` argument as `"prob"`.

```r
titanic_predictions <- dt_titanic |>
  predict(new_data = titanic_test)

titanic_predictions
```

```
# A tibble: 442 x 1
   .pred_class
   <fct>
 1 no
 2 yes
 3 no
 4 no
 5 no
 6 no
 7 no
 8 yes
 9 no
10 no
# ... with 432 more rows
```

If you want to calculate the predicted probabilities, it is necessary to assign the `type` argument as `"prob"`.

```r
dt_titanic |>
  predict(new_data = titanic_test,
          type     = "prob")
```

```
# A tibble: 442 x 2
   .pred_no .pred_yes
      <dbl>     <dbl>
 1   0.828      0.172
 2   0.0822     0.918
 3   0.828      0.172
 4   0.828      0.172
 5   0.828      0.172
 6   0.828      0.172
 7   0.828      0.172
 8   0.0822     0.918
 9   0.828      0.172
10   0.828      0.172
# ... with 432 more rows
```

Evaluating model performance:

`{yardstick}` package is used to evaluate/measure the model performance. Its functions require a data.frame or tibble with model results. To combine the model prediction and actual/observed values of target variable in test data, `cbind()` function can be used as follows:

```r
titanic_results <- tibble(predicted = titanic_predictions$.pred_class,
                          actual    = titanic_test$survived)
```

Then we can calculate the RMSE of the model by using `rmse()` function with obligatory arguments: `truth` must be assigned with actual values, `estimate` must be assigned with predicted values of target variable.

```r
titanic_results |> conf_mat(truth    = actual,
                            estimate = predicted)
```

```
          Truth
Prediction  no yes
       no  299  47
       yes  22  74
```

The `accuracy` metric can be calculated in similar manner by `accuracy()` function:

```r
titanic_results |> accuracy(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.844
```

The `sensitivity` metric can be calculated in similar manner by `sens()` function:

```r
titanic_results |> sens(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.931
```

```r
titanic_results |> spec(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.612
```

`{tidymodels}` ecosystem provides many binary classification metrics:

- `accuracy()`
- `kap()`
- `sens()`
- `spec()`
- `ppv()`
- `npv()`
- `mcc()`
- `j_index()`
- `bal_accuracy()`
- `detection_prevalence()`
- `precision()`
- `recall()`
- `f_meas()`

Streaming model fitting by `last_fit()` function. It takes a model specification, model formula, and data split object.

```r
titanic_last_fit <- dt_model |>
  last_fit(survived ~., split = titanic_split)
```

Collecting metrics:

```r
titanic_last_fit |> collect_metrics()
```

```
# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary         0.844 Preprocessor1_Model1
2 roc_auc  binary         0.818 Preprocessor1_Model1
```

Collecting predictions:

```r
titanic_last_fit |> collect_predictions()
```

```
# A tibble: 442 x 7
   id               .pred_no .pred_yes  .row .pred_class survived .config
   <chr>               <dbl>     <dbl> <int> <fct>       <fct>    <chr>
 1 train/test split   0.828      0.172     3 no          no       Preprocessor1~
 2 train/test split   0.0822     0.918    21 yes         yes      Preprocessor1~
 3 train/test split   0.828      0.172    22 no          no       Preprocessor1~
 4 train/test split   0.828      0.172    28 no          no       Preprocessor1~
 5 train/test split   0.828      0.172    42 no          no       Preprocessor1~
 6 train/test split   0.828      0.172    43 no          no       Preprocessor1~
 7 train/test split   0.828      0.172    47 no          no       Preprocessor1~
 8 train/test split   0.0822     0.918    50 yes         yes      Preprocessor1~
 9 train/test split   0.828      0.172    53 no          no       Preprocessor1~
10 train/test split   0.828      0.172    57 no          no       Preprocessor1~
# ... with 432 more rows
```

Model validation

```r
set.seed(123)
titanic_folds <- vfold_cv(titanic_train,
                          v = 10)
titanic_folds
```

```
#  10-fold cross-validation
# A tibble: 10 x 2
   splits             id
   <list>             <chr>
 1 <split [1588/177]> Fold01
 2 <split [1588/177]> Fold02
 3 <split [1588/177]> Fold03
 4 <split [1588/177]> Fold04
 5 <split [1588/177]> Fold05
 6 <split [1589/176]> Fold06
 7 <split [1589/176]> Fold07
 8 <split [1589/176]> Fold08
 9 <split [1589/176]> Fold09
10 <split [1589/176]> Fold10
```

```r
titanic_wf <- workflow() |>
  add_model(dt_model) |>
```

```r
  add_formula(survived ~.)
```

```r
titanic_fit_cv <- titanic_wf |>
  fit_resamples(titanic_folds)
```

You can see the mean values of metrics over folds:

```r
titanic_fit_cv |> collect_metrics()
```

```
# A tibble: 2 x 6
  .metric  .estimator  mean     n std_err .config
  <chr>    <chr>      <dbl> <int>   <dbl> <chr>
1 accuracy binary     0.793    10 0.00672 Preprocessor1_Model1
2 roc_auc  binary     0.761    10 0.0110  Preprocessor1_Model1
```

Or you can check the metric values for each fold:

```r
titanic_fit_cv |> collect_metrics(summarize = FALSE)
```

```
# A tibble: 20 x 5
   id     .metric  .estimator .estimate .config
   <chr>  <chr>    <chr>          <dbl> <chr>
 1 Fold01 accuracy binary         0.774 Preprocessor1_Model1
 2 Fold01 roc_auc  binary         0.688 Preprocessor1_Model1
 3 Fold02 accuracy binary         0.774 Preprocessor1_Model1
 4 Fold02 roc_auc  binary         0.783 Preprocessor1_Model1
 5 Fold03 accuracy binary         0.808 Preprocessor1_Model1
 6 Fold03 roc_auc  binary         0.775 Preprocessor1_Model1
 7 Fold04 accuracy binary         0.814 Preprocessor1_Model1
 8 Fold04 roc_auc  binary         0.779 Preprocessor1_Model1
 9 Fold05 accuracy binary         0.797 Preprocessor1_Model1
10 Fold05 roc_auc  binary         0.798 Preprocessor1_Model1
11 Fold06 accuracy binary         0.778 Preprocessor1_Model1
12 Fold06 roc_auc  binary         0.715 Preprocessor1_Model1
13 Fold07 accuracy binary         0.818 Preprocessor1_Model1
14 Fold07 roc_auc  binary         0.776 Preprocessor1_Model1
15 Fold08 accuracy binary         0.767 Preprocessor1_Model1
16 Fold08 roc_auc  binary         0.746 Preprocessor1_Model1
17 Fold09 accuracy binary         0.778 Preprocessor1_Model1
```

```
18 Fold09 roc_auc  binary         0.783 Preprocessor1_Model1
19 Fold10 accuracy binary         0.824 Preprocessor1_Model1
20 Fold10 roc_auc  binary         0.765 Preprocessor1_Model1
```