Tidymodels

JOE OYUGI

5/12/2020

##we use nycflights13, skimr and tidymodels packages.

```
library(tidymodels)      # for the recipes package, along with the rest of tidymodels Helper packages
```

```
## -- Attaching packages ------------------------------------ tidymodels 0.1.0 --
```

```
## v broom     0.5.6      v recipes   0.1.10
## v dials     0.0.6      v rsample   0.0.6
## v dplyr     0.8.5      v tibble    3.0.1
## v ggplot2   3.3.0      v tune      0.1.0
## v infer     0.5.1      v workflows 0.1.1
## v parsnip   0.1.0      v yardstick 0.0.6
## v purrr     0.3.4
```

```
## -- Conflicts --------------------------------------- tidymodels_conflicts() --
## x purrr::discard()  masks scales::discard()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x ggplot2::margin() masks dials::margin()
## x recipes::step()   masks stats::step()
```

```
library(nycflights13)    # for flight data
library(skimr)           # for variable summaries
```

##check the data

```
dim(flights)
```

```
## [1] 336776     19
```

```
names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

##let's use the nycflights13 data to predict whether a plane arrives more than 30 minutes late

```
set.seed(123)
flight_data <-
  flights %>%
  mutate(
   #convert the arrival delay to a factor
    arr_delay = ifelse(arr_delay >= 30, "late", "on_time"),
    arr_delay = factor(arr_delay),
```

```
   #we use the date (not date-time) in the recipe below
    date = as.Date(time_hour)
  ) %>%
  #Include the weather data
  inner_join(weather, by = c("origin", "time_hour")) %>%
  #Only retain the specific columns we will use
  select(dep_time, flight, origin, dest, air_time, distance,
        carrier, date, arr_delay, time_hour) %>%
  #Exclude missing data
  na.omit() %>%
  #convert qualitative columns encoded as character to factors
  mutate_if(is.character, as.factor)
```

##inspect the data

```
str(flight_data)
```

```
## tibble [325,819 x 10] (S3: tbl_df/tbl/data.frame)
##  $ dep_time : int [1:325819] 517 533 542 544 554 554 555 557 557 558 ...
##  $ flight   : int [1:325819] 1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ origin   : Factor w/ 3 levels "EWR","JFK","LGA": 1 3 2 2 3 1 1 3 2 3 ...
##  $ dest     : Factor w/ 104 levels "ABQ","ACK","ALB",..: 44 44 58 13 5 69 36 43 54 69 ...
##  $ air_time : num [1:325819] 227 227 160 183 116 150 158 53 140 138 ...
##  $ distance : num [1:325819] 1400 1416 1089 1576 762 ...
##  $ carrier  : Factor w/ 16 levels "9E","AA","AS",..: 12 12 2 4 5 12 4 6 4 2 ...
##  $ date     : Date[1:325819], format: "2013-01-01" "2013-01-01" ...
##  $ arr_delay: Factor w/ 2 levels "late","on_time": 2 2 1 2 2 2 2 2 2 2 ...
##  $ time_hour: POSIXct[1:325819], format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

##16% of the flights in this data set arrived more than 30 minutes late

```
flight_data %>%
  count(arr_delay) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 2 x 3
##   arr_delay      n  prop
##   <fct>      <int> <dbl>
## 1 late       52540 0.161
## 2 on_time   273279 0.839
```

```
##arr_delay is a factor variable then we have flight, a numeric value, and time_hour, a date-time value
##that are retained in the model to be used as identification variables which can be used to troublesho
##predicted data points.
flight_data %>%
  skimr::skim(dest, carrier)
```

Table 1: Data summary

Name	Piped data
Number of rows	325819
Number of columns	10
Column type frequency:	
factor	2

Table 1: Data summary

Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
dest	0	1	FALSE	104	ATL: 16771, ORD: 16507, LAX: 15942,
carrier	0	1	FALSE	16	UA: 57489, B6: 53715, EV: 50868, DL: 4
##there are 104	flight desti	nations containe	d in dest	and 16 dist	inct carriers.
##we are using a	simple logi	stic regression	model, the	variables	dest and carrier will be converted to dun
##variables.					
##we use rsample	package to	create an object	that cont	ains the in	formation on how to split the data, and
##two more rsamp	le functions	to create data	frames for	the traini	ng and testing sets.

###fix the random numbers by setting seed to enable reproducibility when random numbers are used.

```
set.seed(555)
###3/4 of the data are put into the training set
data_split <- initial_split(flight_data, prop = 3/4)
###create data frames for the two sets
train_data <- training(data_split)
test_data <- testing(data_split)
```

##inspection

```
dim(train_data)
```

```
## [1] 244365     10
```

```
dim(test_data)
```

```
## [1] 81454    10
```

##create a recipe to be used to create a few new predictors and some preprocessing required by the model.

```
flights_rec <-
  recipe(arr_delay ~ ., data = train_data)
```

##We add roles to the recipe using update_role() since flight and time hour are variables with a custom role that we called "ID". They won't be used as either outcomes or predictors.

```
flights_rec <-
  recipe(arr_delay ~ ., data = train_data) %>%
  update_role(flight, time_hour, new_role = "ID")
```

##current set of variables and roles

```
summary(flights_rec)
```

```
## # A tibble: 10 x 4
##    variable  type    role      source
##    <chr>     <chr>   <chr>     <chr>
##  1 dep_time  numeric predictor original
##  2 flight    numeric ID        original
##  3 origin    nominal predictor original
##  4 dest      nominal predictor original
```

```
##  5 air_time  numeric predictor original
##  6 distance  numeric predictor original
##  7 carrier   nominal predictor original
##  8 date      date    predictor original
##  9 time_hour date    ID        original
## 10 arr_delay nominal outcome   original
```

##the date column has an R date object so including that column as is will mean that the model will convert it to a numeric format equal to the number of days after a reference date.

```
flight_data %>%
  distinct(date) %>%
  mutate(numeric_date = as.numeric(date))
```

```
## # A tibble: 364 x 2
##    date       numeric_date
##    <date>            <dbl>
##  1 2013-01-01        15706
##  2 2013-01-02        15707
##  3 2013-01-03        15708
##  4 2013-01-04        15709
##  5 2013-01-05        15710
##  6 2013-01-06        15711
##  7 2013-01-07        15712
##  8 2013-01-08        15713
##  9 2013-01-09        15714
## 10 2013-01-10        15715
## # ... with 354 more rows
```

##possibly numeric date variable is a good option for modeling. It is better to add model terms derived from the date that have a better potential to be important to the model, like from the single date variable we could derive the day of the week, the month, whether or not the date corresponds to a holiday. ##Adding them to the recipe

```
flights_rec <-
  recipe(arr_delay ~ ., data = train_data) %>%
  update_role(flight, time_hour, new_role = "ID") %>%
  step_date(date, features = c("dow", "month")) %>%
  step_holiday(date, holidays = timeDate::listHolidays("US")) %>%
  step_rm(date)
```

##use step_dummy() which has all_nominal and -all_outcomes selectors to create dummy variables for all of the factor or character columns unless they are outcomes.

```
flights_rec <-
  recipe(arr_delay ~ ., data = train_data) %>%
  update_role(flight, time_hour, new_role = "ID") %>%
  step_date(date, features = c("dow", "month")) %>%
  step_holiday(date, holidays = timeDate::listHolidays("US")) %>%
  step_rm(date) %>%
  step_dummy(all_nominal(), -all_outcomes())
```

##carrier and dest have some infrequently occurring values, it is possible that dummy variables might be created for values that don't exist in the training set. ##for example there is one destination that is only in the test set

```
test_data %>%
  distinct(dest) %>%
```

```
  anti_join(train_data)
```

```
## Joining, by = "dest"
```

```
## # A tibble: 1 x 1
##   dest
##   <fct>
## 1 LEX
```

##when the recipe is applied to the training set, a column is made for LEX but it will contain all zeros. ##this is a "zero-variance predictor" that has no information within the column ##step_zv() will remove columns from the data when the training set data have a single value, so it is added to the recipe after step_dummy()

```
flights_rec <-
  recipe(arr_delay ~ ., data = train_data) %>%
  update_role(flight, time_hour, new_role = "ID") %>%
  step_date(date, features = c("dow", "month")) %>%
  step_holiday(date, holidays = timeDate::listHolidays("US")) %>%
  step_rm(date) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors())
```

##For modelling we start by building a model specification using the parsnip package.

```
lr_mod <-
  logistic_reg() %>%
  set_engine("glm")
```

##we use a model workflow, which pairs a model and recipe together. ##we use the workflows package from tidymodels to bundle our parsnip model(lr_mod) with our recipe (flights_rec).

```
flights_wflow <-
  workflow() %>%
  add_model(lr_mod) %>%
  add_recipe(flights_rec)
```

```
flights_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: logistic_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 5 Recipe Steps
##
## * step_date()
## * step_holiday()
## * step_rm()
## * step_dummy()
## * step_zv()
##
## -- Model -----------------------------------------------------------------------
## Logistic Regression Model Specification (classification)
##
## Computational engine: glm
```

##Now there is a single function that can be used to prepare the recipe and train the model from the

resulting predictors:

```
flights_fit <-
  flights_wflow %>%
  fit(data = train_data)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have column names if `.name_repair` is omitte
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

##to extract model objects we use pull_workflow_fit() ##to extract recipe objects we use pull_workflow_recipe()

##pulling the fitted model object then use the broom::tidy() function to get a tidy tibble of model coefficients:

```
flights_fit %>%
  pull_workflow_fit() %>%
  tidy()
```

```
## # A tibble: 157 x 5
##    term                          estimate std.error statistic  p.value
##    <chr>                            <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)                    3.91    2.73           1.43 1.51e- 1
##  2 dep_time                      -0.00167 0.0000141   -118.   0.
##  3 air_time                      -0.0439  0.000561     -78.4  0.
##  4 distance                       0.00686 0.00150        4.57 4.84e- 6
##  5 date_USChristmasDay            1.12    0.173          6.49 8.45e-11
##  6 date_USColumbusDay             0.474   0.159          2.99 2.81e- 3
##  7 date_USCPulaskisBirthday       0.864   0.139          6.21 5.47e-10
##  8 date_USDecorationMemorialDay   0.279   0.110          2.53 1.15e- 2
##  9 date_USElectionDay             0.696   0.169          4.12 3.82e- 5
## 10 date_USGoodFriday              1.28    0.166          7.71 1.27e-14
## # ... with 147 more rows
```

##use the trained workflow(flights_fit) to predict with the unseen test data. ##the goal was to predict whether a plane arrives more than 30 minutes late. ##predict() method applies the recipe to the new data, then passes them to the fitted model.

```
predict(flights_fit, test_data)
```

```
## # A tibble: 81,454 x 1
##    .pred_class
##    <fct>
##  1 on_time
##  2 on_time
##  3 on_time
##  4 on_time
##  5 on_time
##  6 on_time
##  7 on_time
##  8 on_time
##  9 on_time
## 10 on_time
## # ... with 81,444 more rows
```

##if we want the predicted class probabilities for each flight we specify type = "prob" in predict(). ##we bind the ouptput with some variables from the test data and save them together.

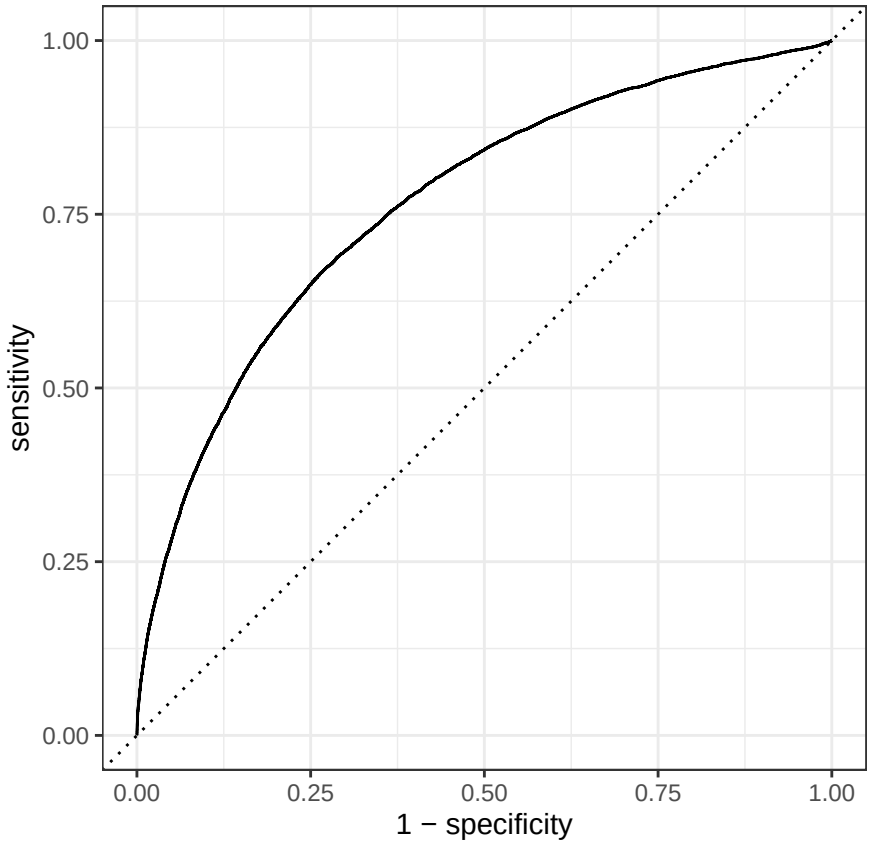
```
flights_pred <-
  predict(flights_fit, test_data, type = "prob") %>%
  bind_cols(test_data %>% select(arr_delay, time_hour, flight))
```

```
head(flights_pred, 10)
```

```
## # A tibble: 10 x 5
##    .pred_late .pred_on_time arr_delay time_hour           flight
##         <dbl>         <dbl> <fct>     <dttm>               <int>
##  1     0.0565         0.944 on_time   2013-01-01 05:00:00   1714
##  2     0.0264         0.974 on_time   2013-01-01 06:00:00     79
##  3     0.0481         0.952 on_time   2013-01-01 06:00:00    301
##  4     0.0325         0.967 on_time   2013-01-01 06:00:00     49
##  5     0.0711         0.929 on_time   2013-01-01 06:00:00   1187
##  6     0.0583         0.942 on_time   2013-01-01 06:00:00   4401
##  7     0.0171         0.983 on_time   2013-01-01 06:00:00   1895
##  8     0.0458         0.954 on_time   2013-01-01 06:00:00    135
##  9     0.0221         0.978 on_time   2013-01-01 06:00:00   4646
## 10     0.0502         0.950 on_time   2013-01-01 06:00:00   4144
```

##to valuate the performance of our workflow we calculate a metric that tells how well our model predicted late arrivals, compared to the true status of our outcome variable, arr_delay. ##we use the area under the ROC curve as our metric, computed using roc_curve() and roc_auc() from the yardstick package.

```
flights_pred %>%
  roc_curve(truth = arr_delay, .pred_late) %>%
  autoplot()
```



##roc_auc() estimates the area under the curve

```
flights_pred %>%
  roc_auc(truth = arr_delay, .pred_late)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.765
```