Classification-TidyModels

March 27, 2022

1 Classification using TidyModels

In this lab we would be going through: - Logistic Regression - Linear Discriminant Analysis - Quadratic Discriminant Analysis

using TidyModels.

For this lab, we would examining the `OJ` data set that contains a number of numeric variables plus a variable called `Purchase` which has the two labels `CH` and `MM` (which is Citrus Hill or Minute Maid Orange Juice)

[107]:
```
suppressPackageStartupMessages(library(tidymodels))
suppressPackageStartupMessages(library(ISLR))
suppressPackageStartupMessages(library(discrim))
suppressPackageStartupMessages(library(corrr))
```

[108]:
```
head(OJ)
```

A data.frame: 6 × 18

	Purchase <fct>	WeekofPurchase <dbl>	StoreID <dbl>	PriceCH <dbl>	PriceMM <dbl>	DiscCH <dbl>	DiscMM <dbl>	Sp <d
1	CH	237	1	1.75	1.99	0.00	0.0	0
2	CH	239	1	1.75	1.99	0.00	0.3	0
3	CH	245	1	1.86	2.09	0.17	0.0	0
4	MM	227	1	1.69	1.69	0.00	0.0	0
5	CH	228	7	1.69	1.69	0.00	0.0	0
6	CH	230	7	1.69	1.99	0.00	0.0	0

[109]:
```
attach(OJ)
```

```
The following objects are masked from OJ (pos = 3):

    DiscCH, DiscMM, ListPriceDiff, LoyalCH, PctDiscCH, PctDiscMM,
    PriceCH, PriceDiff, PriceMM, Purchase, SalePriceCH, SalePriceMM,
    SpecialCH, SpecialMM, STORE, Store7, StoreID, WeekofPurchase
```

The `correlate()` function (from `corrr` package) will calculate the correlation matrix between all the variables that it is being fed.

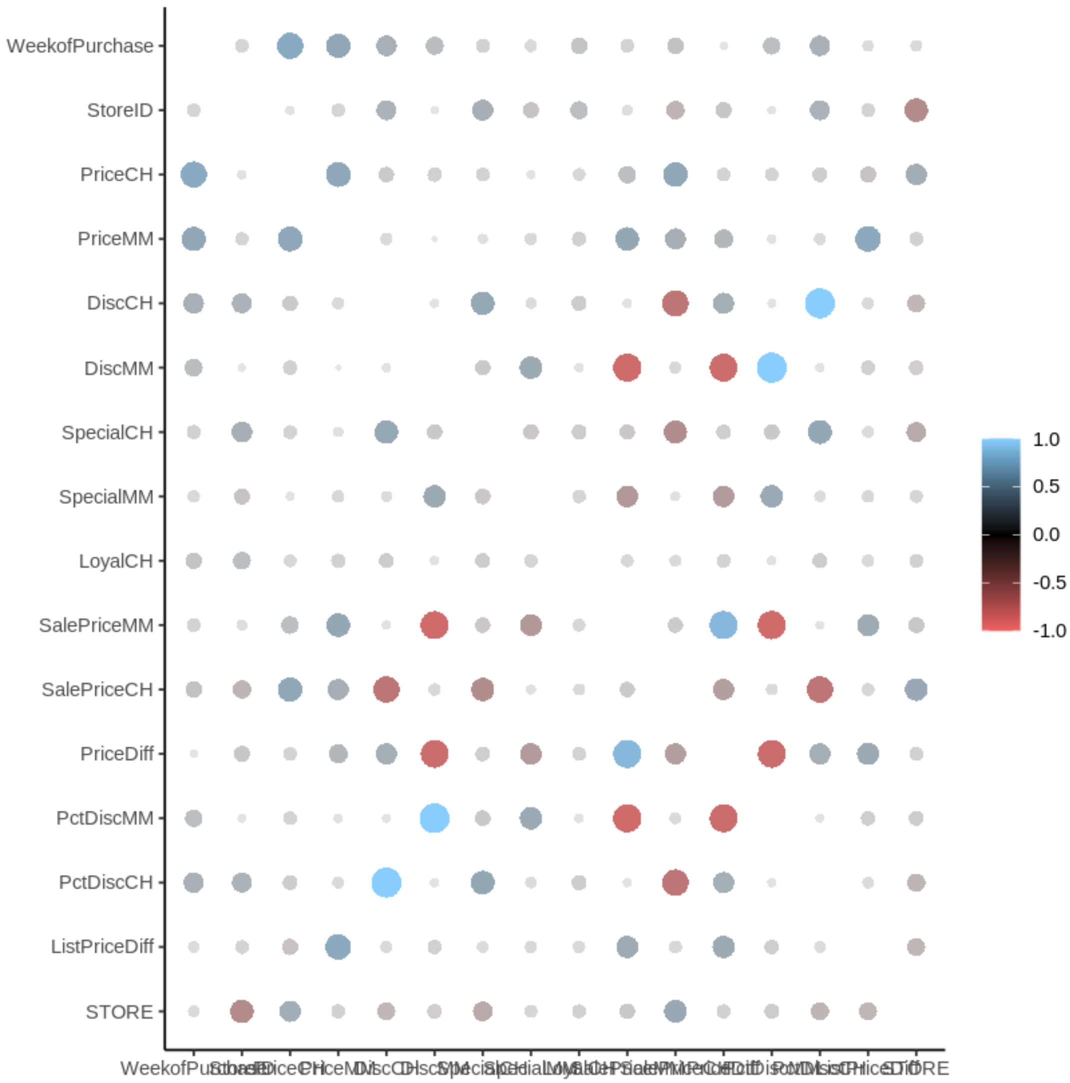
[110]:
```
cor_oj <- OJ %>%
  select(-Purchase, -Store7) %>% #Remove Purchase & Store as it not numeric
  correlate()
```

```
Correlation method: 'pearson'
Missing treated using: 'pairwise.complete.obs'
```

Lets pass this correlation to `rplot()` to visualize the correlation matrix

[111]:
```
rplot(cor_oj, colours = c("indianred2", "black", "skyblue1"))
```

```
Don't know how to automatically pick scale for object of type noquote.
Defaulting to continuous.
```



1.1 Logistic Regression

Now we will fit a logistic regression model. We will again use the **parsnip** package, and we will use `logistic_reg()` to create a logistic regression model specification.

[112]:
```
lr_spec <- logistic_reg() %>%
  set_engine("glm") %>% #default engine
  set_mode("classification") #default mode
```

We want to model the `Direction` of the stock market based on the percentage return from the 5 previous days plus the volume of shares traded.

[113]:
```r
lr_fit <- lr_spec %>%
  fit(
    Purchase ~ PriceCH + PriceMM + SalePriceMM + SalePriceCH + WeekofPurchase,
    data = OJ
    )

lr_fit
```

```
parsnip model object

Fit time:  4ms

Call:  stats::glm(formula = Purchase ~ PriceCH + PriceMM + SalePriceMM +
    SalePriceCH + WeekofPurchase, family = stats::binomial, data = data)

Coefficients:
   (Intercept)         PriceCH         PriceMM     SalePriceMM     SalePriceCH
       2.78191         0.69780        -0.84973        -1.84587         3.49967
WeekofPurchase
      -0.02184

Degrees of Freedom: 1069 Total (i.e. Null);  1064 Residual
Null Deviance:     1431
Residual Deviance: 1320  AIC: 1332
```

[114]:
```r
lr_fit %>%
  pluck("fit") %>%
  summary()
```

```
Call:
stats::glm(formula = Purchase ~ PriceCH + PriceMM + SalePriceMM +
    SalePriceCH + WeekofPurchase, family = stats::binomial, data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7977  -0.9765  -0.7307   1.1663   2.2329

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)     2.78191    1.27287   2.186  0.02885 *
PriceCH         0.69780    1.27510   0.547  0.58420
PriceMM        -0.84973    0.77077  -1.102  0.27027
SalePriceMM    -1.84587    0.32814  -5.625 1.85e-08 ***
SalePriceCH     3.49967    0.79278   4.414 1.01e-05 ***
WeekofPurchase -0.02184    0.00697  -3.134  0.00173 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1430.9  on 1069  degrees of freedom
Residual deviance: 1320.0  on 1064  degrees of freedom
AIC: 1332

Number of Fisher Scoring iterations: 4
```

The `summary()` lets us see a couple of different things such as; parameter estimates, standard errors, p-values, and model fit statistics.

we can use the `tidy()` function to extract some of these model attributes for further analysis or presentation.

[115]:
```
tidy(lr_fit)
```

A tibble: 6 × 5

term <chr>	estimate <dbl>	std.error <dbl>	statistic <dbl>	p.value <dbl>
(Intercept)	2.78190730	1.27286701	2.185544	2.884896e-02
PriceCH	0.69780075	1.27509476	0.547254	5.842042e-01
PriceMM	-0.84973062	0.77076842	-1.102446	2.702678e-01
SalePriceMM	-1.84586939	0.32814297	-5.625199	1.852947e-08
SalePriceCH	3.49966954	0.79278219	4.414415	1.012835e-05
WeekofPurchase	-0.02184087	0.00696997	-3.133568	1.726951e-03

[116]:
```
predict(lr_fit, new_data = OJ)
```

A tibble: 1070 × 1

.pred_class
<fct>
CH
MM
CH
MM
MM
CH
MM
MM
MM
MM
CH
CH
MM
CH
CH
CH
MM
CH
CH
CH
CH
CH
CH
CH
CH
CH
CH
CH
CH
CH
CH
CH
MM
MM
MM
CH
CH
CH
CH
MM
CH
CH
MM
MM
CH
MM
MM
CH
MM
CH

The result is a tibble with a single column `.pred_class` which will be a factor variable of the same labels as the original training data set.

We can also get back probability predictions, by specifying `type = "prob"`

[117]:
```
predict(lr_fit, new_data = OJ, type = "prob")
```

A tibble: 1070 × 2

.pred_CH	.pred_MM
<dbl>	<dbl>
0.6018192	0.39818080
0.4757620	0.52423803
0.7291985	0.27080149
0.4104296	0.58957038
0.4157248	0.58427524
0.6252782	0.37472175
0.4544633	0.54553672
0.4035200	0.59647996
0.4087878	0.59121215
0.4247129	0.57528714
0.5711175	0.42888255
0.8631244	0.13687556
0.4104916	0.58950836
0.7321935	0.26780647
0.5950050	0.40499503
0.5950050	0.40499503
0.4812120	0.51878797
0.7577825	0.24221751
0.7617687	0.23823126
0.6973664	0.30263362
0.7110133	0.28898666
0.7928020	0.20719802
0.7199054	0.28009460
0.9014370	0.09856298
0.7321935	0.26780647
0.8778828	0.12211719
0.7448446	0.25515539
0.7489732	0.25102675
0.8563010	0.14369905
0.5950050	0.40499503
0.6070412	0.3929588
0.6070412	0.3929588
0.4757620	0.5242380
0.4812120	0.5187880
0.3740062	0.6259938
0.7247704	0.2752296
0.7291057	0.2708943
0.7818322	0.2181678
0.7855347	0.2144653
0.4157248	0.5842752
0.6252782	0.3747218
0.6303815	0.3696185
0.4193854	0.5806146
0.4247129	0.5752871
0.5870866	0.4129134
0.4104296	0.5895704
0.4157248	0.5842752
0.6354558	0.3645442
0.3982744	0.6017256
0.6504954	0.3495046

We can describe a `confusion matrix` that would help us understand how well the predictive model is preforming by given a table of predicted values against the true value

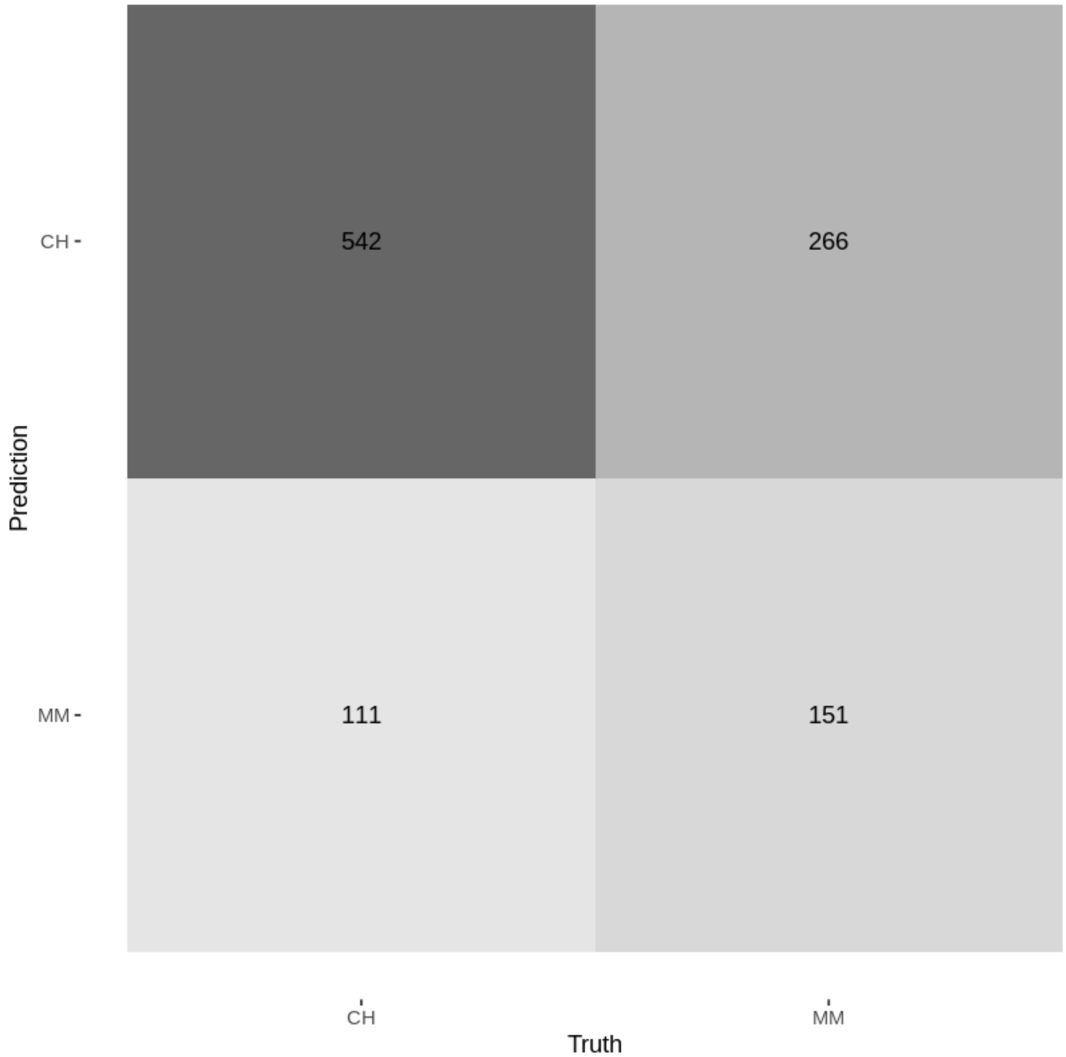
`augment()` function helps add the predictions to the `data.frame` and then use that to look at model performance metrics.

[118]:
```
augment(lr_fit, new_data = OJ) %>%
  conf_mat(truth = Purchase, estimate = .pred_class)
```

```
          Truth
Prediction  CH  MM
        CH 542 266
        MM 111 151
```

We can represent this as a `heatmap`

[119]:
```
augment(lr_fit, new_data = OJ) %>%
  conf_mat(truth = Purchase, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



A good performing model would ideally have high numbers along the diagonal (up-left to down-right) with small numbers on the off-diagonal. We see here that the model isn't great, as it tends to predict "CH" as "MM" more often than it should.

We can also calculate various performance metrics. One of the most common metrics is accuracy, which is how often the model predicted correctly as a percentage.

[120]:
```
augment(lr_fit, new_data = OJ) %>%
  accuracy(truth = Purchase, estimate = .pred_class)
```

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
accuracy	binary	0.6476636

Fitting a model and evaluating the model on the same data would give much information abou the model's performance.

Let us instead split up the data, train it on some of it and then evaluate it on the other part of the data. Since we are working with some data that has a time component,lets train the data over a before a specific week and test it over the set of other weeks.

This would more closely match how such a model would be used in real life.

[121]:
```
mean(WeekofPurchase)
```

254.381308411215

[122]:
```
oj_train <- OJ %>%
  filter(WeekofPurchase < 260)
dim(oj_train)

oj_test <- OJ %>%
  filter(WeekofPurchase >= 260)
dim(oj_test)

dim(OJ)
```

1. 600 2. 18

1. 470 2. 18

1. 1070 2. 18

[141]:
```
# Build an lr model that fits Purchase as response with other numeric variables
# Predictors: PriceCH, PriceMM, DiscCH, DiscMM, PctDiscMM, PctDiscCH
# Modeled over the training data set created above
lr.fit = function(){
    # your code here
    model = lr_spec %>% fit(
        Purchase ~ PriceCH + PriceMM + DiscCH + DiscMM + PctDiscMM + PctDiscCH,
        data = oj_train
        )
    return(model)
}
```

[142]:
```
summary = lr.fit() %>% pluck('fit') %>% summary()
coeff = coef(summary)

stopifnot(round(coeff[1],2) == 1.47) #Intercept test case
stopifnot(round(coeff[2],2) == 2.99) #PriceCH test case
```

[143]:
```
# hidden test cases
```

[144]:
```
# lr.fit = lr_spec %>% fit( Purchase ~ PriceCH + PriceMM + DiscCH + DiscMM +␣
↪PctDiscMM + PctDiscCH, data = oj_train)
```

[147]:
```
# Return a confusion matrix and accuracy of the model lr.fit
# the matrix has to be defined over the test data set
confusion_matrix = function(model=lr.fit()){
    # your code here
    matrix = augment(model, new_data = oj_test) %>% conf_mat(truth = Purchase,␣
 ↪estimate = .pred_class)
    return(matrix)
}

accuracy.fit = function(model=lr.fit()){
    # your code here
    acc = augment(model, new_data = oj_test) %>% accuracy(truth = Purchase,␣
 ↪estimate = .pred_class)
    return(acc)
}
```

[148]:
```
confusion_matrix()


accuracy = accuracy.fit()
stopifnot(round(accuracy[3],2) == 0.70) #Accuracy test case
```

```
          Truth
Prediction  CH  MM
        CH 249  84
        MM  59  78
```

[149]:
```
# hidden test cases
```

1.2 Linear Discriminant Analysis

We will use the `discrim_linear()` function to create a LDA specification. We are gonna use two predictors (`PriceCH` & `PriceMM`) for easy comparision

[131]:
```
lda_spec <- discrim_linear() %>%
  set_mode("classification") %>%
  set_engine("MASS")
```

[132]:
```
lda_fit = lda_spec %>%
  fit(Purchase ~ PriceCH + PriceMM, data = oj_train)

lda_fit
```

```
parsnip model object

Fit time:  2ms
```

```
Call:
lda(Purchase ~ PriceCH + PriceMM, data = data)

Prior probabilities of groups:
   CH    MM
0.575 0.425

Group means:
    PriceCH  PriceMM
CH 1.831333 2.081101
MM 1.816353 2.022980

Coefficients of linear discriminants:
              LD1
PriceCH  7.069496
PriceMM -8.507458
```

[133]:

```
predict(lda_fit, new_data = oj_test)
```

A tibble: 470 × 1

.pred_class
<fct>
CH
CH
CH
MM
MM
CH
CH
CH
CH
CH
CH
CH
CH
MM
CH
CH
CH
MM
MM
MM
MM
MM
MM
MM
MM
MM
MM
MM
MM
MM
CH
CH
CH
CH
CH
CH
CH
CH
CH
CH
MM
MM
CH
CH
CH
MM
MM
MM
CH
CH

[134]:
```
#confusion matrix
augment(lda_fit, new_data = oj_test) %>%
  conf_mat(truth = Purchase, estimate = .pred_class)

#accuracy
augment(lda_fit, new_data = oj_test) %>%
      accuracy(truth = Purchase, estimate = .pred_class)
```

```
          Truth
Prediction  CH  MM
        CH 219 101
        MM  89  61
```

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
accuracy	binary	0.5957447

Lets compare this to `lr()` fit

[135]:
```
lda_fit_2 = lda_spec %>%
  fit(Purchase ~ PriceCH + PriceMM + DiscCH + DiscMM + PctDiscMM + PctDiscCH,
      data = oj_train)

#accuracy
augment(lda_fit_2, new_data = oj_test) %>%
      accuracy(truth = Purchase, estimate = .pred_class)
```

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
accuracy	binary	0.6957447

1.3 Quadratic Discriminant Analysis

We can fit a `QDA` model by using the `discrim_quad()` function.

[136]:
```
qda_spec = discrim_quad() %>%
  set_mode("classification") %>%
  set_engine("MASS")
```

`qda_spec` has a similar usage as `lda_spec`. so,

[137]:
```
qda_fit = qda_spec %>% fit(Purchase ~ PriceCH + PriceMM,
                           data = oj_train)
qda_fit
```

```
parsnip model object

Fit time:  2ms
```

```
Call:
qda(Purchase ~ PriceCH + PriceMM, data = data)

Prior probabilities of groups:
   CH    MM
0.575 0.425

Group means:
    PriceCH  PriceMM
CH 1.831333 2.081101
MM 1.816353 2.022980
```

[138]:
```
#confusion matrix
augment(qda_fit, new_data = oj_test) %>%
  conf_mat(truth = Purchase, estimate = .pred_class)

#accuracy
augment(qda_fit, new_data = oj_test) %>%
  accuracy(truth = Purchase, estimate = .pred_class)
```

```
          Truth
Prediction  CH  MM
        CH 308 162
        MM   0   0
```

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
accuracy	binary	0.6553191

We can see that, `QDA` performs better compared to `LDA` using two predictors

Now lets compare all the three fits with 6 predictors

[139]:
```
qda_fit_2 = qda_spec %>%
  fit(Purchase ~ PriceCH + PriceMM + DiscCH + DiscMM + PctDiscMM + PctDiscCH,
      data = oj_train)

#accuracy
augment(qda_fit_2, new_data = oj_test) %>%
      accuracy(truth = Purchase, estimate = .pred_class)
```

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
accuracy	binary	0.7361702

[140]:
```
get_accuracy = function(fit){
    accuracy = augment(fit, new_data = oj_test) %>%
      accuracy(truth = Purchase, estimate = .pred_class)
```

```
    accuracy[3]
}
accuracy_matrix = matrix(c( get_accuracy(lr.fit())
                           , get_accuracy(lda_fit_2)
                           , get_accuracy(qda_fit_2))
                           , nrow = 1, ncol = 3, byrow=FALSE)
colnames(accuracy_matrix) = c("LR", "LDA", "QDA")
accuracy_matrix
```

```
    Error in lr.fit(): could not find function "lr.fit"
Traceback:


    1. matrix(c(get_accuracy(lr.fit()), get_accuracy(lda_fit_2),
↪get_accuracy(qda_fit_2)),
 .      nrow = 1, ncol = 3, byrow = FALSE)

    2. get_accuracy(lr.fit())

    3. augment(fit, new_data = oj_test) %>% accuracy(truth = Purchase,
 .      estimate = .pred_class)    # at line 2-3 of file <text>

    4. accuracy(., truth = Purchase, estimate = .pred_class)

    5. augment(fit, new_data = oj_test)
```

we can see that the `QDA` works better for this data where as `LDA` and `LR` work similary for this data set