Regression, Part 2

Will Doyle

Introduction

In this section we're going to continue fitting regressions to the training data and testing the predictions against the testing data. We'll include additional continuous variables. We're also going to add some new elements. In particular, We'll be using independent variables or predictor variables that are binary or categorical.

We'll need the same libraries as last week:

```
library(tidyverse)
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'hms'
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## Registered S3 method overwritten by 'tune':
##   method                   from
##   required_pkgs.model_spec parsnip
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.1.4 --
```

```
## v broom        0.7.10     v rsample      0.1.0
## v dials        0.0.10     v tune         0.1.6
## v infer        1.0.0      v workflows    0.2.4
## v modeldata    0.1.1      v workflowsets 0.1.0
## v parsnip      0.1.7      v yardstick    0.0.8
## v recipes      0.2.0
```

```
## Warning: package 'recipes' was built under R version 4.1.2
```

```
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
```

```
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

And the same dataset, which includes data on movies released since 1980.

```
mv<-readRDS("mv.Rds")%>%
  filter(!is.na(budget))%>%
  mutate(log_gross=log(gross))%>%
  mutate(bechdel_bin=ifelse(bechdel_score==3,1,0))%>%
  mutate(bechdel_factor=recode_factor(bechdel_bin,
                                      `1`="Pass",
                                      `0`="Fail",
                                      ))
```

Training and Testing Data

As before, I'm going to split the data into training and testing versions.

```
set.seed(35202)

split_data<-mv%>%initial_split()

mv_train<-training(split_data)

mv_test<-testing(split_data)
```

Set Model

We're going to use the same linear model as last time, so I'll specify that here.

```
lm_fit <-
  linear_reg() %>%
  set_engine("lm")%>%
  set_mode("regression")
```

Begin Workflow

Now I start my workflow. A workflow is pretty much what it sounds like. It's a set of steps in a modeling process. To start with, let's add our model as defined above to the workflow.

```
movie_wf<-workflow()%>%
  add_model(lm_fit)
```

A Brief Digression: The Bechdel Test

The Bechdel test was first made famous by Alison Bechdel in 1985– Bechdel credited the idea to Liz Wallace and her reading of Virginia Woolf. It asks three questions about a movie:

1. Does it have two women in it?
2. Who talk to each other?
3. About something other than a man?

The test sets an unbelievably low bar, and yet a remarkable number of movies don't pass it. One excuse sometimes used by filmmakers is that movie audiences tend to be young and male, and so favor movies that don't necessarily pass this test. However, a study by CAA and shift7 called this logic into question:

A study indicates that female-led movies make more money thatn those that are not.

And here's the study.

Let's see if we can replicate their results in this data. First of all, what proportion of these movies made since 2000 pass the Bechdel test?

```
mv%>%
  group_by(bechdel_bin)%>%
  count()
```

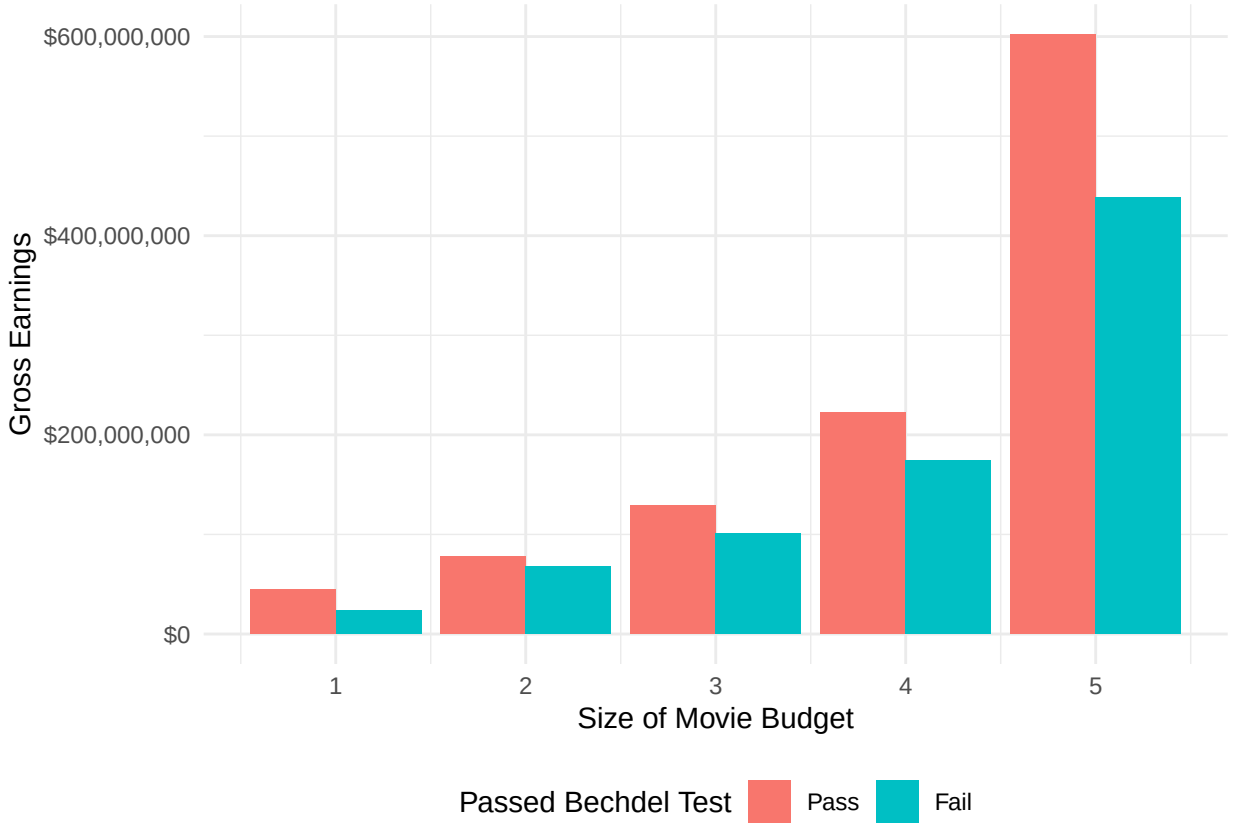
```
## # A tibble: 3 x 2
## # Groups:   bechdel_bin [3]
##   bechdel_bin     n
##         <dbl> <int>
## 1           0   873
## 2           1  1186
## 3          NA  1132
```

A majority, but 873 (873!!) movies did not have two female characters that spoke to each other about anything other than a man.

Let's see if the contention of movie execs about earning power holds up.

```
mv%>%
  mutate(budget_level=ntile(budget,n=5))%>%
  group_by(budget_level,bechdel_factor)%>%
  summarize(mean_gross=mean(gross,na.rm=TRUE))%>%
  drop_na()%>%
  ggplot(aes(x=budget_level,y=mean_gross,fill=bechdel_factor))+
  geom_col(position="dodge")+
  scale_y_continuous(labels=dollar_format())+
  ylab("Gross Earnings")+xlab("Size of Movie Budget")+
  scale_fill_discrete(name="Passed Bechdel Test")+
  theme_minimal()+
  theme(legend.position = "bottom")
```

```
## `summarise()` has grouped output by 'budget_level'. You can override using the
## `.groups` argument.
```



Nope. At every budget level, movies that pass the Bechdel test make more money, not less.

Regression with a binary variable

Let's see if we can use regression to obtain a similar result. The next variable I want to include is the Bechdel variable, which is a binary variable set to "1" if the movie passes the Bechdel test.

```
mv%>%
  group_by(bechdel_bin)%>%
  summarize(count=n())%>%
  mutate(`Proportion`=count/sum(count))%>%
  arrange(-Proportion)
```

```
## # A tibble: 3 x 3
##   bechdel_bin count Proportion
##         <dbl> <int>      <dbl>
## 1           1  1186      0.372
## 2          NA  1132      0.355
## 3           0   873      0.274
```

Set Formula

Next, I add the variable `bechdel_factor` to the formula.

```
movie_formula<-as.formula("log_gross~budget+score+bechdel_factor")
```

The variable `bechdel_factor` is now added to our formula. But we need to tell the model how to handle this variable, since it is a categorical variable.

Set Recipe

```
movie_rec<-recipe(movie_formula,data=split_data)%>%
  step_log(budget)%>%
  step_dummy(bechdel_factor)
```

The `step_dummy` command will convert a factor variable into a series of dummy (0 or 1) variables. There will always be one fewer dummy variable than levels in the factor. For our two-level factor, R will generate one dummy variable.

Add recipe to workflow

We can now add the recipe to the previously existing workflow.

```
movie_wf<-
  movie_wf%>%
  add_recipe(movie_rec)
```

Fit to training data

Now we can fit the processed data to the training dataset and take a look at the results.

```
movie_wf<-movie_wf%>%
  fit(mv_train)
```

```
## Warning: There are new levels in a factor: NA
```

The `tidy` command allows us to see the coefficients from the model fit to the training data.

```
movie_wf%>%
  extract_fit_parsnip()%>%
  tidy()
```

```
## # A tibble: 4 x 5
##   term                estimate std.error statistic   p.value
##   <chr>                  <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)            0.607    0.460       1.32 1.87e-  1
## 2 budget                 0.923    0.0227     40.7  4.27e-246
## 3 score                  0.255    0.0332      7.66 3.26e- 14
## 4 bechdel_factor_Fail   -0.220    0.0602     -3.66 2.59e-  4
```

So if a movie fails the Bechdel test, then it makes less money, even AFTER controlling for budget and IMDB score. This is really important: we always interpret binary variables as a comparison between the group that is set to 1 (fails Bechdel test) and the group that is set to 0 (passes).

```
movie_wf%>%
  extract_fit_parsnip()%>%
  glance()
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic   p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>     <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.523         0.522  1.16      562. 1.30e-246     3 -2410. 4830. 4857.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Predict on testing data and calculate rmse

We can add predictions to the testing dataset in the same way we did before.

```
mv_test<-
  movie_wf%>%
  predict(new_data=mv_test)%>%
  rename(.pred3=.pred)%>%
  bind_cols(mv_test)
```

```
## Warning: There are new levels in a factor: NA
```

Calculate RMSE

Calculating rmse works the same as well.

```
mv_test%>%
  rmse(truth=log_gross,estimate=.pred3)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        1.10
```

It seems like including the Bechdel score increased our accuracy, but we need to be really careful: it's not the same dataset! I didn't have complete data on the Bechdel score, so we dropped a bunch of cases that had missing data. When there's missing data, we can't make direct comparisons of RMSE. We're doing the analysis on a new version of the dataset, limited to only those cases with available data. There ARE some limited solutions to this problem, but the bottom line is that missing data is . . . missing.

Regression with a categorical variable

We can also include categorical variables (not just binary variables) in our model using much the same process. Let's see if a movie's MPAA Rating is related to its gross.

What numbers of movies have different ratings?

```
mv%>%
  group_by(rating)%>%
  count()
```

```
## # A tibble: 9 x 2
## # Groups:   rating [9]
##   rating        n
##   <chr>     <int>
## 1 G            54
## 2 NC-17         6
## 3 Not Rated    37
## 4 PG          434
```

```
## 5 PG-13      1249
## 6 R          1392
## 7 TV-MA         2
## 8 Unrated       7
## 9 <NA>         10
```

Set Formula

```
movie_formula<-as.formula("log_gross~budget+score+rating")
```

Set Recipe

Because our formula now includes a variable that is categorical, we need to change our recipe to reflect that. Below, `step_dummy` is applied to the 'rating" variables.

```
movie_rec<-recipe(movie_formula,mv_train)%>%
  step_log(budget)%>%
  step_other(rating, threshold=.005)%>%
  step_dummy(rating)
```

The `step_other` command is used to identify very rare occurences in a given categorical variable. In this case, it's set to recategorize levels that are less than one half of one percent of the overall sample. Having done that, we can feed it forward to the `step_dummy` command to convert the variable into a series of categorical variables. We will again have a set of dummy variables, one for every level of the categorical variable save one. In this case, the excluded or reference category will be rated G movies.

Update workflow with new recipe

Because we've already created the workflow `movie_wf` we can update it with our new recipe, using the command `update_recipe`.

```
movie_wf<-movie_wf%>%
  update_recipe(movie_rec)
```

Fit to training data and look at coefficients and model fit

Now we're ready to fit our linear model. The fit command below tells it to fit the model, with results stored in the object `lm_results`. We can then pipe the lm_results to the `tidy` command to see the coefficients. To see measures of model fit we can use `extract_fit_parsnip` and then pipe those results to the `glance` command.

```
movie_wf<-movie_wf%>%
  fit(mv_train)
```

```
## Warning: There are new levels in a factor: NA
```

```
movie_wf%>%
  extract_fit_parsnip()%>%
  tidy()
```

```
## # A tibble: 8 x 5
##   term             estimate std.error statistic   p.value
```

```
##   <chr>               <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)         1.61     0.440       3.66 2.53e-  4
## 2 budget              0.856    0.0207     41.4  2.65e-282
## 3 score               0.342    0.0259     13.2  1.82e- 38
## 4 rating_Not.Rated   -2.49     0.310      -8.05 1.27e- 15
## 5 rating_PG          -0.276    0.189      -1.46 1.43e-  1
## 6 rating_PG.13       -0.453    0.181      -2.50 1.25e-  2
## 7 rating_R           -0.992    0.183      -5.43 6.11e-  8
## 8 rating_other       -1.40     0.401      -3.50 4.70e-  4
```

Interpreting categorical variables is a bit tricky. What this shows is that "Not Rated" movies have a log gross that is 2.19 lower than rated G movies. Similarly, rated R movies have a log gross that is .875 lower than rated G movies. All of the coefficients from that categorical variable– everything that starts with "rating_" must be interpreted relative to the reference category– rated G movies. The choice of the reference category is arbitrary and can be changed.

```
movie_wf%>%
  extract_fit_parsnip()%>%
  glance()
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.544         0.543  1.19      405.       0     7 -3787. 7592. 7644.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Predict on testing data and calculate rmse

With our new variable included, we can do our normal steps of generating a prediction and adding it to the testing dataset.

```
mv_test<-
  movie_wf%>%
  predict(mv_test)%>%
  rename(.pred4=.pred)%>%
  bind_cols(mv_test)
```

```
## Warning: There are new levels in a factor: NA
```

Calculate RMSE

With the data in the testing dataset, we can then generate the RMSE from our new model.

```
mv_test%>%rmse(truth=log_gross,estimate=.pred4)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        1.26
```

Using `last_fit`

The `tidymodels` package has a function that automates the steps of running the model, generating predictions in the testing dataset and then generating metrics of model fit from the testing dataset. It's called `last_fit`.

This accomplishes the same steps above, but does it all at once.

```
lf<-last_fit(movie_wf,split=split_data)
```

```
## Warning: package 'rlang' was built under R version 4.1.2
```

```
## Warning: package 'vctrs' was built under R version 4.1.2
```

```
## ! train/test split: preprocessor 1/1: There are new levels in a factor: NA
```

```
## ! train/test split: preprocessor 1/1, model 1/1 (predictions): There are new levels in a fac...
```

```
lf$.metrics
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       1.26  Preprocessor1_Model1
## 2 rsq     standard       0.505 Preprocessor1_Model1
```

As you can see we get the same RMSE from last fit as when we did it "by hand."

In class work

Add a categorical and a continuous variable to the dataset and see what happens to model fit in the testing dataset.

New formula

```
movie_formula<-as.formula("log_gross~budget+score+rating+genre+runtime")
```

Change recipe to reflect new variables

```
movie_rec<-recipe(movie_formula,mv_train)%>%
  step_log(budget)%>%
  step_other(rating, genre, threshold=.005)%>%
  step_dummy(rating,genre)
```

Update recipe in workflow

```
movie_wf<-movie_wf%>%
  update_recipe(movie_rec)
```

Fit model

```
movie_wf<-movie_wf%>%
  fit(mv_train)
```

```
## Warning: There are new levels in a factor: NA
```

Take a look at coefficients

```
movie_wf%>%
  extract_fit_parsnip()%>%
  tidy()
```

```
## # A tibble: 17 x 5
##    term              estimate std.error statistic   p.value
##    <chr>                <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)       1.46       0.479       3.05  2.29e-  3
```

```
##  2 budget            0.838      0.0246     34.0   3.97e-207
##  3 score             0.421      0.0297     14.2   6.47e- 44
##  4 runtime           0.000627   0.00183     0.343 7.32e-  1
##  5 rating_Not.Rated -2.49       0.313      -7.95  2.85e- 15
##  6 rating_PG        -0.222      0.187      -1.19  2.35e-  1
##  7 rating_PG.13     -0.412      0.195      -2.11  3.47e-  2
##  8 rating_R         -0.976      0.197      -4.96  7.46e-  7
##  9 rating_other     -1.18       0.400      -2.95  3.22e-  3
## 10 genre_Adventure  -0.175      0.119      -1.48  1.40e-  1
## 11 genre_Animation  -0.0413     0.133      -0.311 7.56e-  1
## 12 genre_Biography  -0.615      0.105      -5.83  6.15e-  9
## 13 genre_Comedy     -0.126      0.0689     -1.82  6.87e-  2
## 14 genre_Crime      -0.537      0.109      -4.95  7.94e-  7
## 15 genre_Drama      -0.405      0.0807     -5.02  5.45e-  7
## 16 genre_Horror      0.948      0.123       7.70  1.92e- 14
## 17 genre_other      -0.217      0.231      -0.940 3.47e-  1
```

Look at model fit

```
lf<-last_fit(movie_wf,split=split_data)
```

```
## ! train/test split: preprocessor 1/1: There are new levels in a factor: NA
```

```
## ! train/test split: preprocessor 1/1, model 1/1 (predictions): There are new levels in a fac...
```

```
lf$.metrics
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       1.23  Preprocessor1_Model1
## 2 rsq     standard       0.534 Preprocessor1_Model1
```

Last Note

Remember that we need to carefully distinguish between categorical variables and continuous variables when including them in our models. If we're using categorical variables we'll need to pre-process the data in order to let the model know that these variables should be included as categorical variables, with an excluded reference category.