**Explore the data**

Our modeling goal here is to predict the IMDB ratings for episodes of *The Office* based on the other characteristics of the episodes in the dataset There are two datasets, one with the ratings and one with information like director, writer, and which character spoke which line. The episode numbers and titles are not consistent between them, so we can use regular expressions to do a better job of matching the datasets up for joining.

library(tidyverse)

ratings\_raw <- readr::read\_csv("<https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-03-17/office_ratings.csv>")

remove\_regex <- "[:punct:]|[:digit:]|parts |part |the |and"

office\_ratings <- ratings\_raw %>%

transmute(

episode\_name = str\_to\_lower(title),

episode\_name = str\_remove\_all(episode\_name, remove\_regex),

episode\_name = str\_trim(episode\_name),

imdb\_rating

)

office\_info <- schrute::theoffice %>%

mutate(

season = as.numeric(season),

episode = as.numeric(episode),

episode\_name = str\_to\_lower(episode\_name),

episode\_name = str\_remove\_all(episode\_name, remove\_regex),

episode\_name = str\_trim(episode\_name)

) %>%

select(season, episode, episode\_name, director, writer, character)

office\_info

## # A tibble: 55,130 x 6

## season episode episode\_name director writer character

##

## 1 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Michael

## 2 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Jim

## 3 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Michael

## 4 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Jim

## 5 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Michael

## 6 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Michael

## 7 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Michael

## 8 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Pam

## 9 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Michael

## 10 1 1 pilot Ken Kwapis Ricky Gervais;Stephen Merch… Pam

## # … with 55,120 more rows

We are going to use several different kinds of features for modeling. First, let’s find out how many times characters speak per episode.

characters <- office\_info %>%

count(episode\_name, character) %>%

add\_count(character, wt = n, name = "character\_count") %>%

filter(character\_count > 800) %>%

select(-character\_count) %>%

pivot\_wider(

names\_from = character,

values\_from = n,

values\_fill = list(n = 0)

)

characters

## # A tibble: 185 x 16

## episode\_name Andy Angela Darryl Dwight Jim Kelly Kevin Michael Oscar Pam

##

## 1 a benihana … 28 37 3 61 44 5 14 108 1 57

## 2 aarm 44 39 30 87 89 0 30 0 28 34

## 3 after hours 20 11 14 60 55 8 4 0 10 15

## 4 alliance 0 7 0 47 49 0 3 68 14 22

## 5 angry y 53 7 5 16 19 13 9 0 7 29

## 6 baby shower 13 13 9 35 27 2 4 79 3 25

## 7 back from v… 3 4 6 22 25 0 5 70 0 33

## 8 banker 1 2 0 17 0 0 2 44 0 5

## 9 basketball 0 3 15 25 21 0 1 104 2 14

## 10 beach games 18 8 0 38 22 9 5 105 5 23

## # … with 175 more rows, and 5 more variables: Phyllis , Ryan ,

## # Toby , Erin , Jan

Next, let’s find which directors and writers are involved in each episode. I’m choosing here to combine this into one category in modeling, for a simpler model, since these are often the same individuals.

creators <- office\_info %>%

distinct(episode\_name, director, writer) %>%

pivot\_longer(director:writer, names\_to = "role", values\_to = "person") %>%

separate\_rows(person, sep = ";") %>%

add\_count(person) %>%

filter(n > 10) %>%

distinct(episode\_name, person) %>%

mutate(person\_value = 1) %>%

pivot\_wider(

names\_from = person,

values\_from = person\_value,

values\_fill = list(person\_value = 0)

)

creators

## # A tibble: 135 x 14

## episode\_name `Ken Kwapis` `Greg Daniels` `B.J. Novak` `Paul Lieberste…

##

## 1 pilot 1 1 0 0

## 2 diversity d… 1 0 1 0

## 3 health care 0 0 0 1

## 4 basketball 0 1 0 0

## 5 hot girl 0 0 0 0

## 6 dundies 0 1 0 0

## 7 sexual hara… 1 0 1 0

## 8 office olym… 0 0 0 0

## 9 fire 1 0 1 0

## 10 halloween 0 1 0 0

## # … with 125 more rows, and 9 more variables: `Mindy Kaling` , `Paul

## # Feig` , `Gene Stupnitsky` , `Lee Eisenberg` , `Jennifer

## # Celotta` , `Randall Einhorn` , `Brent Forrester` , `Jeffrey

## # Blitz` , `Justin Spitzer`

Next, let’s find the season and episode number for each episode, and then finally let’s put it all together into one dataset for modeling.

office <- office\_info %>%

distinct(season, episode, episode\_name) %>%

inner\_join(characters) %>%

inner\_join(creators) %>%

inner\_join(office\_ratings %>%

select(episode\_name, imdb\_rating)) %>%

janitor::clean\_names()

office

## # A tibble: 136 x 32

## season episode episode\_name andy angela darryl dwight jim kelly kevin

##

## 1 1 1 pilot 0 1 0 29 36 0 1

## 2 1 2 diversity d… 0 4 0 17 25 2 8

## 3 1 3 health care 0 5 0 62 42 0 6

## 4 1 5 basketball 0 3 15 25 21 0 1

## 5 1 6 hot girl 0 3 0 28 55 0 5

## 6 2 1 dundies 0 1 1 32 32 7 1

## 7 2 2 sexual hara… 0 2 9 11 16 0 6

## 8 2 3 office olym… 0 6 0 55 55 0 9

## 9 2 4 fire 0 17 0 65 51 4 5

## 10 2 5 halloween 0 13 0 33 30 3 2

## # … with 126 more rows, and 22 more variables: michael , oscar ,

## # pam , phyllis , ryan , toby , erin , jan ,

## # ken\_kwapis , greg\_daniels , b\_j\_novak ,

## # paul\_lieberstein , mindy\_kaling , paul\_feig ,

## # gene\_stupnitsky , lee\_eisenberg , jennifer\_celotta ,

## # randall\_einhorn , brent\_forrester , jeffrey\_blitz ,

## # justin\_spitzer , imdb\_rating

Just for kicks, let’s show one graph.

office %>%

ggplot(aes(episode, imdb\_rating, fill = as.factor(episode))) +

geom\_boxplot(show.legend = FALSE)

**Train a model**

We can start by loading the tidymodels metapackage, and splitting our data into training and testing sets.

library(tidymodels)

office\_split <- initial\_split(office, strata = season)

office\_train <- training(office\_split)

office\_test <- testing(office\_split)

Then, we build a recipe for data preprocessing.

* First, we must tell the recipe() what our model is going to be (using a formula here) and what our training data is.
* Next, we update the role for episode\_name, since this is a variable we might like to keep around for convenience as an identifier for rows but is not a predictor or outcome.
* Next, we remove any numeric variables that have zero variance.
* As a last step, we normalize (center and scale) the numeric variables. We need to do this because it’s important for LASSO regularization.

The object office\_rec is a recipe that has **not** been trained on data yet (for example, the centered and scaling has not been calculated) and office\_prep is an object that **has** been trained on data. The reason I use strings\_as\_factors = FALSE here is that my ID column episode\_name is of type character (as opposed to, say, integers).

office\_rec <- recipe(imdb\_rating ~ ., data = office\_train) %>%

update\_role(episode\_name, new\_role = "ID") %>%

step\_zv(all\_numeric(), -all\_outcomes()) %>%

step\_normalize(all\_numeric(), -all\_outcomes())

office\_prep <- office\_rec %>%

prep(strings\_as\_factors = FALSE)

Now it’s time to **specify** and then **fit** our models. Here I set up one model specification for LASSO regression; I picked a value for penalty (sort of randomly) and I set mixture = 1 for LASSO. I am using a workflow() in this example for convenience; these are objects that can help you manage modeling pipelines more easily, with pieces that fit together like Lego blocks. You can fit() a workflow, much like you can fit a model, and then you can pull out the fit object and tidy() it!

lasso\_spec <- linear\_reg(penalty = 0.1, mixture = 1) %>%

set\_engine("glmnet")

wf <- workflow() %>%

add\_recipe(office\_rec)

lasso\_fit <- wf %>%

add\_model(lasso\_spec) %>%

fit(data = office\_train)

lasso\_fit %>%

pull\_workflow\_fit() %>%

tidy()

## # A tibble: 1,639 x 5

## term step estimate lambda dev.ratio

##

## 1 (Intercept) 1 8.39 0.183 0

## 2 (Intercept) 2 8.39 0.167 0.0332

## 3 jim 2 0.0113 0.167 0.0332

## 4 michael 2 0.0150 0.167 0.0332

## 5 (Intercept) 3 8.39 0.152 0.0640

## 6 jim 3 0.0247 0.152 0.0640

## 7 michael 3 0.0283 0.152 0.0640

## 8 (Intercept) 4 8.39 0.139 0.0986

## 9 dwight 4 0.00236 0.139 0.0986

## 10 jim 4 0.0361 0.139 0.0986

## # … with 1,629 more rows

If you have used glmnet before, this is the familiar output where we can see (here, for the most regularized examples) what contributes to higher IMDB ratings.

**Tune LASSO parameters**

So we fit one LASSO model, but how do we know the right regularization parameter penalty? We can figure that out using resampling and tuning the model. Let’s build a set of bootstrap resamples, and set penalty = tune() instead of to a single value. We can use a function penalty() to set up an appropriate grid for this kind of regularization model.

set.seed(1234)

office\_boot <- bootstraps(office\_train, strata = season)

tune\_spec <- linear\_reg(penalty = tune(), mixture = 1) %>%

set\_engine("glmnet")

lambda\_grid <- grid\_regular(penalty(), levels = 50)

Now it’s time to tune the grid, using our workflow object.

doParallel::registerDoParallel()

set.seed(2020)

lasso\_grid <- tune\_grid(

wf %>% add\_model(tune\_spec),

resamples = office\_boot,

grid = lambda\_grid

)

What results did we get?

lasso\_grid %>%

collect\_metrics()

## # A tibble: 100 x 6

## penalty .metric .estimator mean n std\_err

##

## 1 1.00e-10 rmse standard 0.638 25 0.0177

## 2 1.00e-10 rsq standard 0.0911 25 0.0132

## 3 1.60e-10 rmse standard 0.638 25 0.0177

## 4 1.60e-10 rsq standard 0.0911 25 0.0132

## 5 2.56e-10 rmse standard 0.638 25 0.0177

## 6 2.56e-10 rsq standard 0.0911 25 0.0132

## 7 4.09e-10 rmse standard 0.638 25 0.0177

## 8 4.09e-10 rsq standard 0.0911 25 0.0132

## 9 6.55e-10 rmse standard 0.638 25 0.0177

## 10 6.55e-10 rsq standard 0.0911 25 0.0132

## # … with 90 more rows

That’s nice, but I would rather see a visualization of performance with the regularization parameter.

lasso\_grid %>%

collect\_metrics() %>%

ggplot(aes(penalty, mean, color = .metric)) +

geom\_errorbar(aes(

ymin = mean - std\_err,

ymax = mean + std\_err

),

alpha = 0.5

) +

geom\_line(size = 1.5) +

facet\_wrap(~.metric, scales = "free", nrow = 2) +

scale\_x\_log10() +

theme(legend.position = "none")

This is a great way to see that regularization helps this modeling a lot. We have a couple of options for choosing our final parameter, such as select\_by\_pct\_loss() or select\_by\_one\_std\_err(), but for now let’s stick with just picking the lowest RMSE. After we have that parameter, we can finalize our workflow, i.e. update it with this value.

lowest\_rmse <- lasso\_grid %>%

select\_best("rmse", maximize = FALSE)

final\_lasso <- finalize\_workflow(

wf %>% add\_model(tune\_spec),

lowest\_rmse

)

We can then fit this finalized workflow on our training data. While we’re at it, let’s see what the most important variables are using the vip package.

library(vip)

final\_lasso %>%

fit(office\_train) %>%

pull\_workflow\_fit() %>%

vi(lambda = lowest\_rmse$penalty) %>%

mutate(

Importance = abs(Importance),

Variable = fct\_reorder(Variable, Importance)

) %>%

ggplot(aes(x = Importance, y = Variable, fill = Sign)) +

geom\_col() +

scale\_x\_continuous(expand = c(0, 0)) +

labs(y = NULL)

And then, finally, let’s return to our test data. The tune package has a function last\_fit() which is nice for situations when you have tuned and finalized a model or workflow and want to fit it one last time on your training data and evaluate it on your testing data. You only have to pass this function your finalized model/workflow and your split.

last\_fit(

final\_lasso,

office\_split

) %>%

collect\_metrics()

## # A tibble: 2 x 3

## .metric .estimator .estimate

##

## 1 rmse standard 0.381

## 2 rsq standard 0.234