Prediction + Model validation

Dr. Maria Tackett

10.24.19

[Click for PDF of slides]

Announcements

- Peer Feedback #2 **due TODAY at 11:59p**
- Complete Reading 06 if you haven't already
- Project proposal **due Friday at 11:59p**
- HW 03 **due Thursday, Oct 31 at 11:59p**

Model selection

Data: Candy Rankings

- Take about 10 - 15 minutes to finish/ review your model selection and data visualization
- Make sure all your work is in one RStudio Cloud project
  - Call that project Ultimate Candy Rankings - Model Selection

Packages

```r
library(tidyverse)
library(broom)
library(knitr)
library(modelr)  # new!
```

Full model

What percent of the variability in win percentages is explained by the model?

```
full_model <- lm(winpercent ~ chocolate + fruity + caramel + peanutyalmondy +
                   nougat + crispedricewafer + hard + bar +
                   pluribus + sugarpercent + pricepercent,
                 data = candy_rankings)
glance(full_model)$r.squared
```

```
## [1] 0.5402088
```

```
glance(full_model)$adj.r.squared
```

```
## [1] 0.4709252
```

Akaike Information Criterion

$$ AIC = -2log(L) + 2k $$

- L: likelihood of the model
  - Likelihood of seeing these data given the estimated model parameters
  - Won't go into calculating it in this course (but you will in future courses)
- Used for model selection, lower the better
  - Value is not informative on its own
- Applies a penalty for number of parameters in the model, k
  - Different penalty than adjusted R^2 but similar idea

```
glance(full_model)$AIC
```

```
## [1] 657.2701
```

Model selection -- a little faster

```r
selected_model <- step(full_model, direction = "backward")
```

```r
tidy(selected_model) %>% select(term, estimate) %>%
  kable(format = "markdown", digits = 3)
```

term	estimate
(Intercept)	32.941
chocolateTRUE	19.147
fruityTRUE	8.881
peanutyalmondyTRUE	9.483
crispedricewaferTRUE	8.385
hardTRUE	-5.669
sugarpercent	7.979

Selected variables

variable	selected
chocolate	x
fruity	x
caramel	
peanutyalmondy	x
nougat	
crispedricewafer	x
hard	x
bar	
pluribus	
sugarpercent	x
pricepercent	

Coefficient interpretation

Interpret the slopes of **`chocolate`** and **`sugarpercent`** in context of the data.

term	estimate
(Intercept)	32.941
chocolateTRUE	19.147
fruityTRUE	8.881
peanutyalmondyTRUE	9.483
crispedricewaferTRUE	8.385
hardTRUE	-5.669
sugarpercent	7.979

AIC

As expected, the selected model has a smaller AIC than the full model. In fact, the selected model has the minimum AIC of all possible main effects models.

```
glance(full_model)$AIC
```

```
## [1] 657.2701
```

```
glance(selected_model)$AIC
```

```
## [1] 649.5113
```

Parismony

Look at the variables in the full and the selected model. Can you guess why some of them may have been dropped? Remember: We like parsimonious models.

variable	selected
chocolate	x
fruity	x
caramel	
peanutyalmondy	x
nougat	
crispedricewafer	x
hard	x
bar	
pluribus	
sugarpercent	x
pricepercent	

Model validation

Overfitting

- The data we are using to construct our models come from a larger population.
- Ultimately we want our model to tell us how the population works, not just the sample we have.
- If the model we fit is too tailored to our sample, it might not perform as well with the remaining population. This means the model is "overfitting" our data.
- We measure this using **model validation** techniques.
- Note: Overfitting is not a huge concern with linear models with low level interactions, however it can be with more complex and flexible models. The following is just an example of model validation, even though we're using it in a scenario where the concern for overfitting is not high.

Model validation

- One commonly used model validation technique is to partition your data into training and testing set
- That is, fit the model on the training data
- And test it on the testing data
- Evaluate model performance using $RMSE$, root-mean squared error

$$ RMSE = \sqrt{\frac{\sum_{i = 1}^n (y_i - \hat{y}_i)^2}{n}} $$

Do you think we should prefer low or high RMSE?

Random sampling and reproducibility

Gotta set a seed!

```
set.seed(102319)
```

- Use different seeds from each other
- Need inspiration? https://www.random.org/

Cross validation

More specifically, k-fold cross validation

- Split your data into k folds.
- Use 1 fold for testing and the remaining (k - 1) folds for training.
- Repeat k times.

Aside -- the modulo operator

```r
9 %% 5
```

```
## [1] 4
```

obs	fold
1	1
2	2
3	3
4	4
5	5
6	1
7	2
8	3

```r
(1:8) %% 5
```

```
## [1] 1 2 3 4 0 1 2 3
```

```r
((1:8) - 1) %% 5
```

```
## [1] 0 1 2 3 4 0 1 2
```

```r
(((1:8) - 1) %% 5) + 1
```

```
## [1] 1 2 3 4 5 1 2 3
```

Prepping your data for 5-fold CV

```
candy_rankings_cv <- candy_rankings %>%
  mutate(id = 1:n()) %>%
  sample_n(nrow(candy_rankings)) %>%
  mutate( fold = (((1:n()) - 1) %% 5) + 1 )

candy_rankings_cv %>%
  count(fold)
```

```
## # A tibble: 5 x 2
##    fold     n
##   <dbl> <int>
## 1     1    17
## 2     2    17
## 3     3    17
## 4     4    17
## 5     5    17
```

CV 1

```
test_fold <- 1
test <- candy_rankings_cv %>% filter(fold == test_fold)
train <- candy_rankings_cv %>% anti_join(test, by = "id")
mod <- lm(winpercent ~ chocolate + fruity + peanutyalmondy + crispedricewafer
(rmse_test1 <- rmse(mod, test))
```

```
## [1] 10.2658
```

RMSE on training vs. testing

Would you expect the RMSE to be higher for your training data or your testing data? Why?

RMSE on training vs. testing

RMSE for testing:

```
(rmse_test1 <- rmse(mod, test))
```

```
## [1] 10.2658
```

RMSE for training:

```
(rmse_train1 <- rmse(mod, train))
```

```
## [1] 9.995652
```

CV 2

```r
test_fold <- 2
test <- candy_rankings_cv %>% filter(fold == test_fold)
train <- candy_rankings_cv %>% anti_join(test, by = "id")
mod <- lm(winpercent ~ chocolate + fruity + peanutyalmondy + crispedricewafer
```

```r
(rmse_test2 <- rmse(mod, test))
```

```
## [1] 9.106138
```

```r
(rmse_train2 <- rmse(mod, train))
```

```
## [1] 10.27274
```

CV 3

```
test_fold <- 3
test <- candy_rankings_cv %>% filter(fold == test_fold)
train <- candy_rankings_cv %>% anti_join(test, by = "id")
mod <- lm(winpercent ~ chocolate + fruity + peanutyalmondy + crispedricewafer
```

```
(rmse_test3 <- rmse(mod, test))
```

```
## [1] 10.54619
```

```
(rmse_train3 <- rmse(mod, train))
```

```
## [1] 9.922409
```

CV 4

```
test_fold <- 4
test <- candy_rankings_cv %>% filter(fold == test_fold)
train <- candy_rankings_cv %>% anti_join(test, by = "id")
mod <- lm(winpercent ~ chocolate + fruity + peanutyalmondy + crispedricewafer
```

```
(rmse_test4 <- rmse(mod, test))
```

```
## [1] 10.16521
```

```
(rmse_train4 <- rmse(mod, train))
```

```
## [1] 10.02132
```

CV 5

```
test_fold <- 5
test <- candy_rankings_cv %>% filter(fold == test_fold)
train <- candy_rankings_cv %>% anti_join(test, by = "id")
mod <- lm(winpercent ~ chocolate + fruity + peanutyalmondy + crispedricewafer
```

```
(rmse_test5 <- rmse(mod, test))
```

```
## [1] 10.10826
```

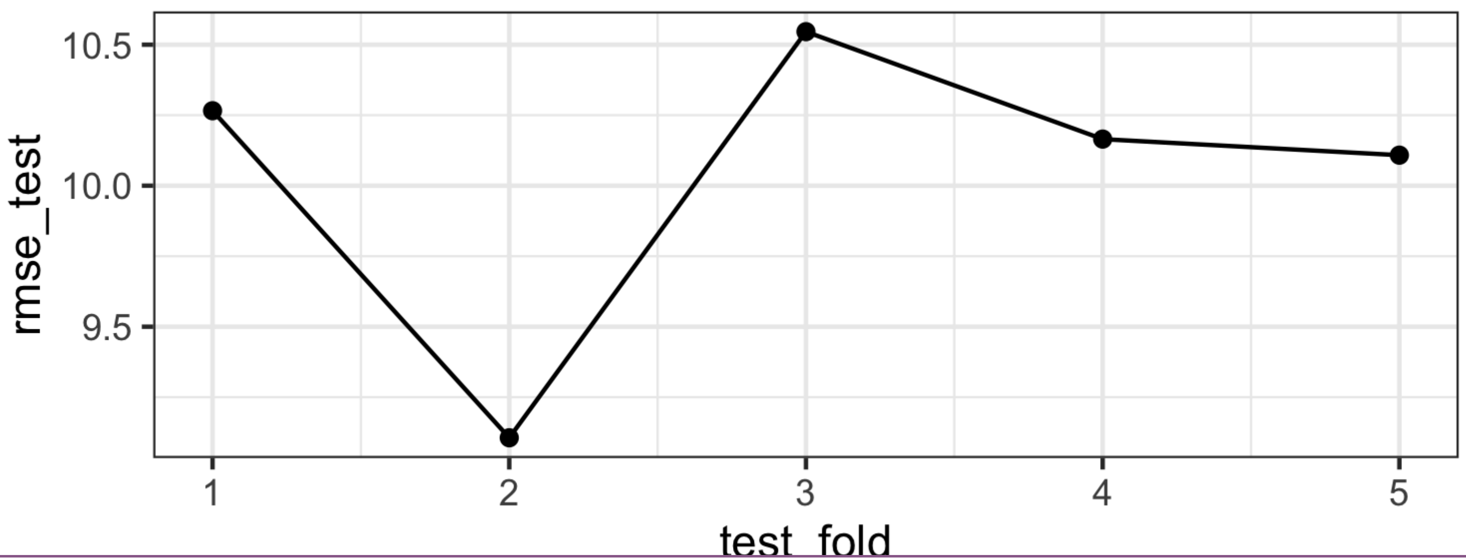
```
(rmse_train5 <- rmse(mod, train))
```

```
## [1] 10.03571
```

Putting it altogether

```
rmse_candy <- tibble(
  test_fold  = 1:5,
  rmse_train = c(rmse_train1, rmse_train2, rmse_train3, rmse_train4, rmse_tra
  rmse_test  = c(rmse_test1, rmse_test2, rmse_test3, rmse_test4, rmse_test5)
)
```

```
ggplot(data = rmse_candy, mapping = aes(x = test_fold, y = rmse_test)) +
  geom_point() +
  geom_line()
```



How does RMSE compare to y?

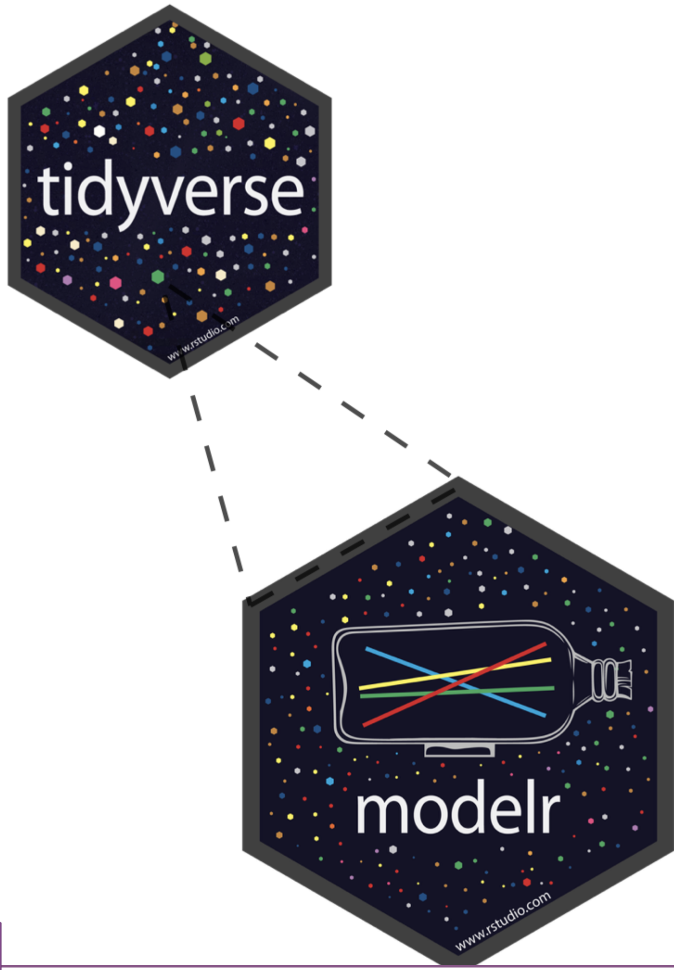
- `winpercent` summary stats:

```
## # A tibble: 1 x 6
##     min   max  mean   med    sd   IQR
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  22.4  84.2  50.3  47.8  14.7  20.7
```

- `rmse_test` summary stats:

```
## # A tibble: 1 x 6
##     min   max  mean   med    sd   IQR
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  9.11  10.5  10.0  10.2 0.548 0.158
```

`modelr` in tidyverse



The **modelr** package provides functions that help you create pipelines when modeling.

```
library(modelr)
```

modelr.tidyverse.org

Cross Validation - Faster

- **`modelr::crossv_kfold`**: Partition data into k folds
- **`purrr::map`**: Fit a model on each of the training sets
- Calculate RMSEs for each of the models on the testing set

Partition data into k folds.

k = 5:

```
candy_rankings_cv <- crossv_kfold(candy_rankings, 5)
candy_rankings_cv
```

```
## # A tibble: 5 x 3
##   train          test           .id
##   <named list> <named list> <chr>
## 1 <resample>   <resample>   1
## 2 <resample>   <resample>   2
## 3 <resample>   <resample>   3
## 4 <resample>   <resample>   4
## 5 <resample>   <resample>   5
```

Fit model on each of training set

```
models <- map(candy_rankings_cv$train, ~
              lm(winpercent ~ chocolate + fruity + peanutyalmondy +
                   crispedricewafer + hard + sugarpercent,
                 data =.))
```

Calculate RMSEs

Explain how **`map2_dbl`** works.

```
rmses <- map2_dbl(models, candy_rankings_cv$test, rmse)
rmses
```

```
##         1         2         3         4         5
## 10.877690  9.373646 10.881654  7.517380 13.240856
```

Prediction

New observation

To make a prediction for a new observation we need to create a data frame with that observation.

Suppose we want to make a prediction for a candy that contains chocolate, isn't fruity, has no peanuts or almonds, has a wafer, isn't hard, and has a sugar content in the 20th percentile.

The following will result in an incorrect prediction. Why? How would you correct it?

```
new_candy <- tibble(chocolate = TRUE,
                    fruity = FALSE,
                    peanutyalmondy = FALSE,
                    crispedricewafer = TRUE,
                    hard = FALSE,
                    sugarpercent = 20)
```

New observation, corrected

```
new_candy <- tibble(chocolate = TRUE,
                    fruity = FALSE,
                    peanutyalmondy = FALSE,
                    crispedricewafer = TRUE,
                    hard = FALSE,
                    sugarpercent = 0.20)
```

Prediction

```
predict(selected_model, newdata = new_candy)
```

```
##        1
## 62.06853
```

Uncertainty around prediction

- Confidence interval around $\hat{y}$ for new data (average win percentage for candy types with the given characteristics):

```
predict(selected_model, newdata = new_candy, interval = "confidence")
```

```
##        fit      lwr      upr
## 1 62.06853 53.65186 70.48519
```

- Prediction interval around $\hat{y}$ for new data (predicted score for an individual type of candy with the given characteristics):

```
predict(selected_model, newdata = new_candy, interval = "prediction")
```

```
##        fit      lwr      upr
## 1 62.06853 39.54943 84.58762
```