Week 9: Model Performance and Feature Engineering

03/24/2020

Jake Campbell

Measuring Model Performance

- The main purpose of most predictive modeling problems is to make predictions
- It makes sense, than, to measure model performance off of our predictions
    - Using residuals with linear regression and predicted classes and probabilities with logistic regression

Remember Overfitting

- We'll be looking at predictions made on our training data today
- These predictions are likely going to be better than we should actually expect
    - Our model has seen this data before
- Next class, we'll be going over cross-validation, allowing us to see how our model performs on new data

Linear Regression Model Metrics

- MAE: Mean Absolute Error
    - Takes the average absolute value of your residuals

```
mpg_output %>%
  mae(truth = hwy, estimate = .fitted)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mae     standard        2.16
```

Linear Regression Model Metrics

- MSE: Mean Square Error
    - Takes the average of squared residuals
- RMSE: Root Mean Square Error
    - Takes the square root of the MSE

```
mean(mpg_output$.resid^2)
```

```
## [1] 8.619549
```

```
mpg_output %>%
  rmse(truth = hwy, estimate = .fitted)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        2.94
```

Logistic Regression Metrics

- Accuracy:

$$\frac{AllCorrectPredictions}{AllPossiblePredictions}$$

- True Positives and Negatives: predictions that were correct from the positive and negative class, respectively
- False Positives and Negatives: predictions that were predicted in the wrong class(positive and negative respectively)

Logistic Regression Metrics

- Sensitivity:

$$\frac{TP}{TP + FN}$$

  - Accuracy rate of the positive class

```
options(yardstick.event_first = FALSE)
biopsy_output %>%
  sens(truth = class, estimate = .fitted_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.921
```

Logistic Regression Metrics

- Specificity:

$$\frac{TN}{TN + FP}$$

  - Accuracy rate of the negative class

```
options(yardstick.event_first = FALSE)
biopsy_output %>%
  spec(truth = class, estimate = .fitted_class)
```

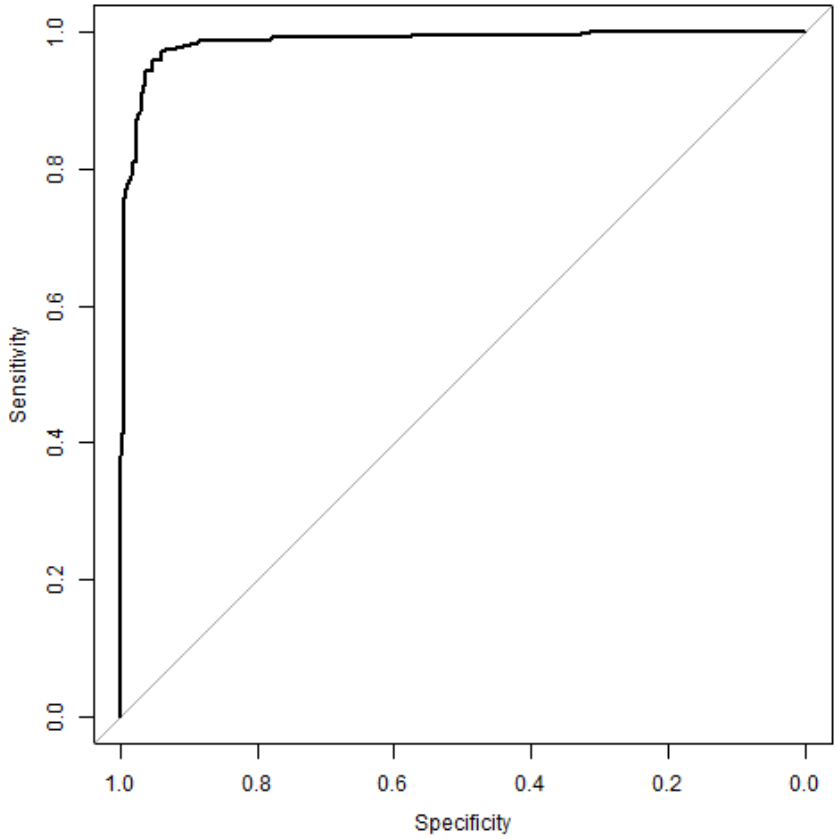
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary         0.965
```

ROC and AUC

- ROC (Receiver Operating Characteristic) is a curve that plots the sensitivity and specificity at different predictive threshholds
  - By threshhold, we mean how to split our predictive probabilities into classes
  - Raising the threshhold increases our specificity and vice versa
- AUC (Area Under the Curve) is the area under the ROC curve
  - The closer AUC is to 1, the better the classifier

ROC and AUC

```
roc(response = biopsy_output$class,
    predictor = biopsy_output$.fitted, plot = T)
```




```
##
## Call:
## roc.default(response = biopsy_output$class, predictor = biopsy_output$.fitted,     plot = T)
```

Feature Engineering

- A lot of the time, predictors are not laid out in front of us
- We have to transform some features to have them actually be useful in our model
    - Ex: a specific date doesn't hold much predictive power, but a month could

Splitting Text

- `str_split` can be used to split text off of some pattern
  - `str_split_fixed` returns a matrix instead of a list

```
transmission <- str_split_fixed(string = mpg$trans, pattern = "[(]", n = 2)
head(transmission)
```

```
##      [,1]     [,2]
## [1,] "auto"   "l5)"
## [2,] "manual" "m5)"
## [3,] "manual" "m6)"
## [4,] "auto"   "av)"
## [5,] "auto"   "l5)"
## [6,] "manual" "m5)"
```

Replacing Text

- `gsub` can be used to replace a certain character in text with another
  - A lot of times, you may want to replace a character with nothing, in which case, you would replace it with `""`

```
mpg <- mpg %>%
  mutate(price = gsub(pattern = "[$]", replacement = "", price)) %>%
  mutate(price = as.numeric(price))
```

Dealing with Dates

- Dates are a specific data type in R
- We can use `as.Date` to turn something into a date
  - We need to specify the `format` argument to say what order the date is in
  - By default, the format is `"%Y-%m-%d"` for full year, numeric month, and numeric day
  - For example: `2019-12-01`

Dealing with Dates

- `format` is also a function we can use to pull a specific part of a date object out

```
mpg <- mpg %>%
  mutate(month = format(x = production_date, "%m"),
         year = format(x = production_date, "%Y"),
         day = format(x = production_date, "%d"))
```