

Lab 10 Solutions

Stats 32: Introduction to R for Undergraduates

Harrison Li

5/2/2024

```
library(tidyverse)

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.4      v readr      2.1.5
## v forcats    1.0.0      v stringr    1.5.1
## v ggplot2    3.4.4      v tibble     3.2.1
## v lubridate  1.9.3      v tidyr      1.3.1
## v purrr      1.0.2
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

set.seed(2024) # For reproducibility
```

Evaluating predictive performance

We fit various regression models, and evaluate their mean squared error.

```
nrow(mpg)
```

```
## [1] 234
```

1. Split the `mpg` dataset **randomly** into a training set of 154 observations and a test set of 80 observations. That is, randomly select 154 of the observations to be in the training dataset (storing it in a variable called `train`), and assign the remaining 80 observations to a variable called `test`. Hint: generate the training indices randomly from `1:nrow(mpg)` using `sample()` and store them in a variable called `train_ind`. Then index appropriately into the `mpg` data frame (it may help to recall negative indexing).

Answer:

```
train_ind <- sample(1:nrow(mpg), size=154, replace=FALSE)
train <- mpg[train_ind,]
test <- mpg[-train_ind,]
```

2. Fit the linear model `hwy ~ cty` using the training data.

Answer:

```
model <- lm(formula=hwy ~ cty, data=train)
```

Recall the definition of mean squared error. If \hat{y} is a vector of n predictions from a model, while y is a vector of the n corresponding true values the model is trying to predict, then the mean squared error is the average of the entries of $(y - \hat{y})^2$. Here's some code to illustrate this (we use `runif()` from Lab 9 to

generate random uniform data for `y_hat` and `y` in this example, but in real life `y` would be in your data and `y_hat` would be an output from your model):

```
y_hat <- runif(n=50)
y <- runif(n=50)
mse <- mean((y-y_hat)^2)
mse
```

```
## [1] 0.2688858
```

3. Use the `predict()` function to get a vector of predictions on the test set, using the linear model fit on the training data in the previous question. Compute the mean squared error on the test set. Is this a measure of in-sample or out-of-sample error?

Answer:

```
y_pred <- predict(object=model, newdata=test)
test_mse <- mean((test$hwy - y_pred)^2)
test_mse
```

```
## [1] 3.550843
```

Since we are computing the mean squared error on a dataset that was not used for fitting the model, we are measuring out-of-sample error.

Now we use `geom_smooth()` to look at the predictions of two loess models — fit to the training dataset — designed to predict `hwy` from `cty`. One has `span = 0.2`, the other with `span = 0.8`.

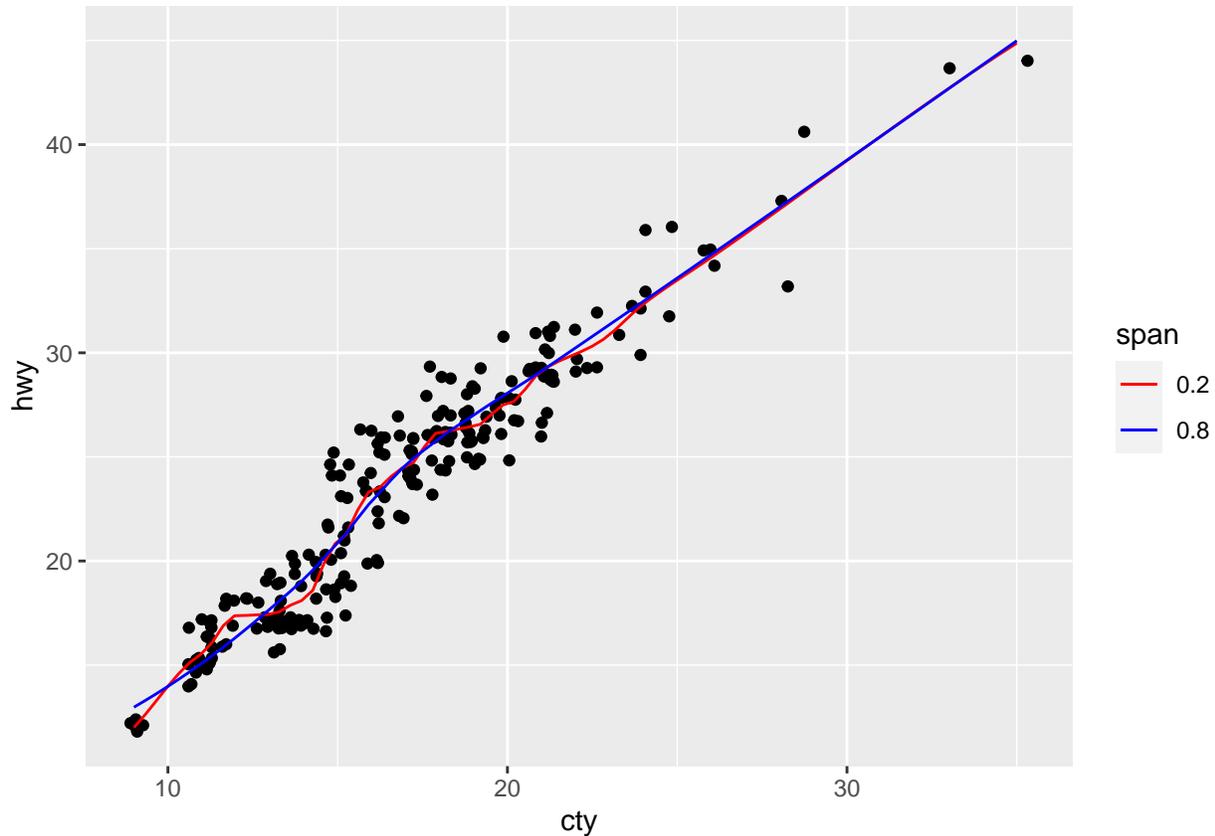
```
mpg %>%
  ggplot(aes(x=cty, y=hwy)) +
  geom_jitter() +
  geom_smooth(method="loess", formula=y ~ x, span=0.2, aes(color="0.2"), lwd=0.5, se=FALSE) +
  geom_smooth(method="loess", formula=y ~ x, span=0.8, aes(color="0.8"), lwd=0.5, se=FALSE) +
  scale_color_manual(name="span",
                     breaks=c("0.2", "0.8"),
                     values=c("0.2"="red", "0.8"="blue"))
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : pseudoinverse used at 16
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : neighborhood radius 1
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : There are other near singularities as well. 1
```



4. Which of the loess models has lower mean squared error on the training set? How about on the test set? Which model would you prefer for prediction? Do the loess models perform better than the linear model from question 3? Hint: you need to first re-fit the models manually (since `geom_smooth()` does not store the model objects it creates under the hood). Note: You may need to ignore NA predictions for evaluating the test MSE.

Answer:

```
# Fit models
loess_wiggly <- loess(formula=hwy ~ cty, data=train, span=0.2)

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : pseudoinverse used at 16
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : neighborhood radius 1
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : reciprocal condition number 0
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : There are other near singularities as well. 1
loess_smooth <- loess(formula=hwy ~ cty, data=train, span=0.8)

# Evaluate train and test mse
loess_wiggly_train_mse <- mean((predict(loess_wiggly, newdata=train)-train$hwy)^2)
loess_smooth_train_mse <- mean((predict(loess_smooth, newdata=train)-train$hwy)^2)
loess_wiggly_test_mse <- mean((predict(loess_wiggly, newdata=test)-test$hwy)^2, na.rm=TRUE)
loess_smooth_test_mse <- mean((predict(loess_smooth, newdata=test)-test$hwy)^2, na.rm=TRUE)
```

```
# Print train mse's  
loess_wiggly_train_mse
```

```
## [1] 2.202707
```

```
loess_smooth_train_mse
```

```
## [1] 2.539745
```

```
# Print test mse's  
loess_wiggly_test_mse
```

```
## [1] 3.122013
```

```
loess_smooth_test_mse
```

```
## [1] 2.716025
```

The wiggly loess model with `span=0.2` has lower MSE than the model with `span=0.8` on the training set but noticeably higher MSE on the test set. This suggests the model with `span=0.2` may be overfitting, and we prefer the model with `span=0.8` based on the test error estimate.

The test MSE of both loess models is also noticeably lower than that of the linear model (see question 3), suggesting we are indeed getting better performance than linear regression.