MACS 33002: Homework #2

Xin Feng

Due Monday, Feb 3rd by 5pm

```
## -- Attaching packages ---------------------------------------------------------------
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------------------------------------------- ti
## x dplyr::combine() masks gridExtra::combine()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## Loading required package: lattice
## Loading required package: ggformula
## Loading required package: ggstance
##
## Attaching package: 'ggstance'
## The following objects are masked from 'package:ggplot2':
##
##     geom_errorbarh, GeomErrorbarh
##
## New to ggformula?  Try the tutorials:
##  learnr::run_tutorial("introduction", package = "ggformula")
##  learnr::run_tutorial("refining", package = "ggformula")
## Loading required package: mosaicData
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
## Registered S3 method overwritten by 'mosaic':
##   method                           from
```

```
##   fortify.SpatialPolygonsDataFrame ggplot2
##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.
##
## Attaching package: 'mosaic'
## The following object is masked from 'package:Matrix':
##
##     mean
## The following objects are masked from 'package:dplyr':
##
##     count, do, tally
## The following object is masked from 'package:purrr':
##
##     cross
## The following object is masked from 'package:ggplot2':
##
##     stat
## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var
## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
##
## Attaching package: 'modelr'
## The following object is masked from 'package:broom':
##
##     bootstrap
## The following object is masked from 'package:mosaic':
##
##     resample
## The following object is masked from 'package:ggformula':
##
##     na.warn
## Loading required package: carData
##
```

```
## Attaching package: 'car'

## The following objects are masked from 'package:mosaic':
##
##     deltaMethod, logit

## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

##
## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':
##
##     nasa

## dummies-1.5.6 provided by Decision Patterns

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

##
## Attaching package: 'rcfss'

## The following object is masked from 'package:modelr':
##
##     mse

## For binary classification, the first factor level is assumed to be the event.
## Set the global option `yardstick.event_first` to `FALSE` to change this.

##
## Attaching package: 'yardstick'

## The following objects are masked from 'package:modelr':
##
##     mae, mape, rmse

## The following object is masked from 'package:readr':
##
##     spec
```

The 2008 NES Data

The `nes2008.csv` data contains a paired down selection of features from the full 2008 American National Election Studies survey. These data will allow you to test competing factors that may influence attitudes towards Joe Biden. The variables are coded as follows:

- `biden` - feeling thermometer ranging from 0-100. Feeling thermometers are a common metric in survey research used to gauge attitudes or feelings of "warmth" towards individuals and institutions. They range from 0-100, with 0 indicating extreme "coldness" and 100 indicating extreme "warmth."
- `female` - 1 if respondent is female, 0 if respondent is male
- `age` - age of respondent in years
- `educ` - number of years of formal education completed by respondent
- `dem` - 1 if respondent is a Democrat, 0 otherwise
- `rep` - 1 if respondent is a Republican, 0 otherwise

For this exercise we consider the following functional form,

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon,$$

where Y is the Joe Biden feeling thermometer, and $[X_1 \dots X_p]$ are the predictive features, including age, gender, education, Democrat, and Republican. The reason for including both `dem` and `rep` party affiliation features is to allow for capturing the preferences of Independents, which must be left out to serve as the baseline category, otherwise we would encounter perfect multicollinearity.

The Questions

1. (10 points) Estimate the MSE of the model using the traditional approach. That is, fit the linear regression model using the *entire* dataset and calculate the mean squared error for the *entire* dataset. Present and discuss your results at a simple, high level.

```
defaultDataDir = "/Users/Liz/Desktop/problem-set-2-master"
fileName = "nes2008.csv"
fileLocation = file.path(defaultDataDir, fileName)
biden_full = read.csv(file = fileLocation, header = T, na.strings = "?")
#Fit the linear model
biden_fu <- lm(biden ~ female + age + educ + dem + rep, data = biden_full)

#Calculate the MSE
(full_mse <- augment(biden_fu, newdata = biden_full) %>%
  mse(truth = biden, estimate = .fitted))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mse     standard        395.
```

Mean Square Error is simply the averaged squared errors found between the actual data points and the model fitted values. With MSE = 395, the error is pretty large in our model; this means that our linear model may not be the best fitted one for our data. Also, there might be outliers that influence the MSE value.

2. (30 points) Calculate the test MSE of the model using the simple holdout validation approach.
   - (5 points) Split the sample set into a training set (50%) and a holdout set (50%). **Be sure to set your seed prior to this part of your code to guarantee reproducibility of results.**

```
biden_full <- as_tibble(biden_full)

set.seed(124)

biden_split <- initial_split(data = biden_full,
                             prop = 0.5)
test = testing(biden_split)
train = training(biden_split)
```

   - (5 points) *Fit* the linear regression model using *only* the *training* observations.

```
biden_tr <- lm(biden ~ female + age + educ + dem + rep, data = train)
```

   - (10 points) Calculate the *MSE* using *only* the *test* set observations.

```
(test_mse <- augment(biden_tr, newdata = test) %>%
  mse(truth = biden, estimate = .fitted))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mse     standard        392.
```

   - (10 points) How does this value compare to the training MSE from question 1? Present numeric comparison and discuss a bit.

Question 2's MSE = 392, which is similar to the 395 in question 1. This implies that model in Q1 doesn't suffer from overfitting. However, we should also notice that there is randomness involved in splitting trainng and testing dataset. If we get a large MSE in Q2, that simply means the testing dataset differs a lot from the training dataset. Thus, the training model is not a good prediction of the testing data.

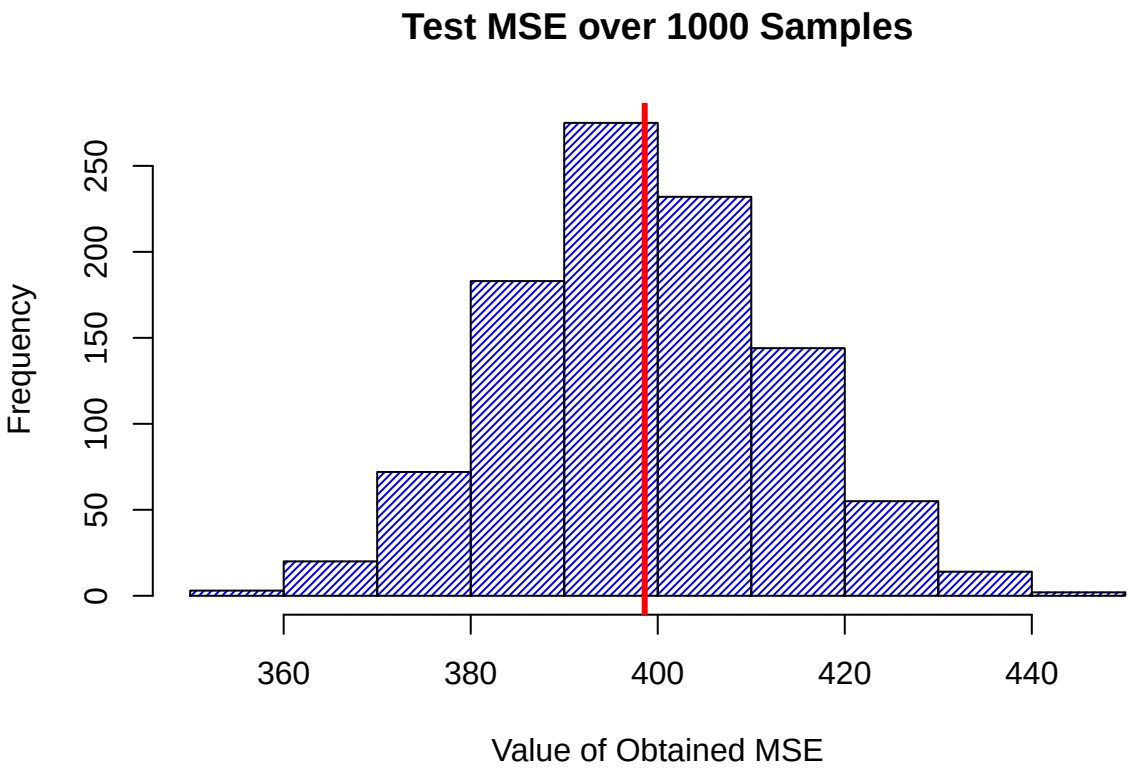
3. (30 points) Repeat the simple validation set approach from the previous question 1000 times, using 1000 different splits of the observations into a training set and a test/validation set.

Visualize your results as a sampling distribution (hint: think histogram or density plots). Comment on the results obtained.

```
set.seed(5)

test_mse=c()

for(i in 1:1000){
  train = sample(1:nrow(biden_full),0.5*nrow(biden_full))
  test = setdiff(1:nrow(biden_full),train)
  biden_tr <- lm(biden ~ female + age + educ + dem + rep, data = biden_full[train, ])
  preds = predict(biden_tr,newx = biden_full[test, ])
  test_mse[i] = mean((biden_full$biden-predict(biden_tr, biden_full))[test]^2)
}

mean(test_mse)
```

```
## [1] 398.6
```

```
hist(test_mse, density=35, main = "Test MSE over 1000 Samples", xlab = "Value of Obtained M
abline(v=mean(test_mse), lwd=3, col="red")
```



The mean of the MSEs from 1000 different combinations of training and testing set is 398.6. Notice that our distribution of 1000 MSEs looks quite like a normal distribution and the centern is roughly at the mean 398.6. 398.6 is also close to our Q1's MSE = 395. This tells us that Q1's MSE is a good measure of uncertainty of the given linear model.

4. (30 points) Compare the estimated parameters and standard errors from the original model in question 1 (the model estimated using *all of the available data*) to parameters and standard errors estimated using the bootstrap ($B = 1000$). Comparison should include, at a minimum, both numeric output as well as discussion on differences, similarities, etc. Talk also about the conceptual use and impact of bootstrapping.

```
#Estimated parameters and SE from Original Model
tidy(biden_fu)
```

```
## # A tibble: 6 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  58.8       3.12       18.8  2.69e-72
## 2 female        4.10      0.948       4.33 1.59e- 5
## 3 age           0.0483    0.0282      1.71 8.77e- 2
## 4 educ         -0.345     0.195      -1.77 7.64e- 2
## 5 dem          15.4       1.07       14.4  8.14e-45
## 6 rep         -15.8       1.31      -12.1  2.16e-32
```

```
# bootstrapped estimates of the parameter estimates and standard errors
lm_coefs <- function(splits, ...) {
  mod <- lm(..., data = analysis(splits))
  tidy(mod)
}

biden_boot <- biden_full %>%
  bootstraps(1000) %>%
  mutate(coef = map(splits, lm_coefs, as.formula(biden ~ female + age + educ + dem + rep)))

biden_boot %>%
  unnest(coef) %>%
  group_by(term) %>%
  summarize(.estimate = mean(estimate),
            .se = sd(estimate, na.rm = TRUE))
```

```
## # A tibble: 6 x 3
##   term        .estimate    .se
##   <chr>           <dbl>  <dbl>
## 1 (Intercept)   58.8    2.92
## 2 age            0.0480 0.0289
## 3 dem           15.4    1.10
## 4 educ          -0.342  0.191
```

```
## 5 female         4.11   0.963
## 6 rep          -15.8    1.37
```

For the original model in question 1:

$$\hat{Y_1} = 58.811 + 4.103 * female + 0.048 * age + -0.345 * educ + 15.424 * dem + -15.850 * rep + \epsilon,$$
$$SE_{\beta_0} = 3.124, SE_{female} = 0.948, SE_{age} = 0.028, SE_{educ} = 0.195, SE_{dem} = 1.068, SE_{rep} = 1.311$$

For the bootstrapping model:

$$\hat{Y_1} = 58.954 + 4.082 * female + 0.047 * age + -0.352 * educ + 15.465 * dem + -15.783 * rep + \epsilon,$$
$$SE_{\beta_0} = 3.068, SE_{female} = 0.923, SE_{age} = 0.029, SE_{educ} = 0.194, SE_{dem} = 1.106, SE_{rep} = 1.423$$

Bootstrapping is one of the most widely used tools to estimate measure of uncertainty associated with a given statistical method. Here, we are using bootstrapping to estimate uncertainty around estimators. As we seen from above, the bootstrapping method generate a set of standard error that is pretty close to the original model. This tells us that our original model's estimators are closed to the true values of the population. Also, bootstrapping gives your similar coefficients as the population regression because it is drawing repeated samples with replacement.