Intro to Machine Learning - Problem Set 2: Uncertainty, Holdouts, and Bootstrapping

Ipek Cinar
2/2/2020

Joe Biden and Validation

Joe Biden was the 47th Vice President of the United States. He was the subject of many memes, attracted the attention of Leslie Knope, and experienced a brief surge in attention due to photos from his youth.
The goal here is to fit a regression model predicting feelings toward Biden, and then implement a couple validation techniques to evaluate the original findings. The validation techniques include the simple holdout approach and the bootstrap. **Note**: we are *not* covering cross validation (LOOCV or k-fold) in this problem set, as these topics are covered in the following week.

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

Question 1

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.

```
# Setting the seed
set.seed(1234)
# Constructing the linear model
lm_model <- lm(biden ~ female + age + educ + dem + rep, data = biden_data)
sum_model <- summary(lm_model)
# Calculating the MSE
entire_mse <- modelr::mse(lm_model, biden_data)
# The function below would be another way to calculate the MSE,
# and mse & mse2 produces the same result.
# mse2 <- function(sum_model)
    # mean(sum_model$residuals^2)
```

Mean squared error (MSE) is the measure denoting the average of the squares of the errors. (The squaring is necessary to remove any negative signs/values so that negative and positive values do not cancel each other) In other words, MSE is the value of the average squared difference between the estimated and actual values, allowing us to evaluate the fit of the model to the data by looking at how close the observed data points are to the model's predicted values. Since, the larger the number (MSE) –> the larger the error, the bigger the value of MSE –> the worse the model is. Hence, we would like to minimize the MSE. We usually use MSE to compare different estimators: by measuring and comparing the values of the different estimators' MSE values amongst each other. The smaller the value of MSE, the better. Therefore, there is no absolute "correct value" for MSE. Since there is no correct answer, the MSE's value relies in allowing us to select one prediction model over another. (It is important to note that however, since due to the nature of it calculation, the MSE puts more weight to larger differences; even if we make a single very bad prediction, the squaring of the error will make the error even worse and may influence the metric towards overestimating the model's badness. In contrast, if all the errors are small, than the opposite effect will prevail: we may then indeed underestimate the model's badness.)

MSE is useful in evaluating the machine learning algorithms. While we say that the smaller the MSE, the better the fit of the model to our data, in machine learning, very small MSE may mean that the data is being overfit by your estimator. In other words, the ideal MSE is not always equal 0, since then we would have a model that perfectly predicts your training data, but which is very unlikely to perfectly predict any other data. What we want is a balance between overfitting (very low MSE for training data) and underfitting (very high MSE for test/validation/unseen data).

Question 2

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.** * (5 points) *Fit* the linear regression model using *only* the *training* observations. * (10 points) Calculate the *MSE* using *only* the *test* set observations. * (10 points) How does this value compare to the training MSE from question 1? Present numeric comparison and discuss a bit.

```
# Setting the seed
set.seed(1234)
# Split the data into two sets using the simple holdout validation approach.
biden_split <- initial_split(data = biden_data, prop = 0.5)
# Training set:
biden_train <- training(biden_split)
# Test set:
biden_test <- testing(biden_split)
# Fitting the linear regression model using only the training observations:
lm_train_model <- lm(biden ~ female + age + educ + dem + rep,
                     data = biden_train)
# summary(lm_train_model)
# Calculating the MSE using only the test set observations:
test_mse <- augment(lm_train_model, newdata = biden_test) %>%
  rcfss::mse(truth = biden, estimate = .fitted) %>%
  select(.estimate)%>%as.numeric()
test_mse
```

```
## [1] 389.1582
```

In addition to the reasons outlined in the discussion provided within the first question, for the estimator to be a good one, a small MSE is better since it implies better agreement between the prediction and the reality (accuracy). Mean squared error (MSE) is also widely used as a loss function (known as squared error loss), we can also use the loss intrepretation. In line with the goal of minimizing MSE, we would also like to minimize the loss function, hence the lesser the value of MSE –> the smaller the error we get from the estimator/parameter/model –> the better the estimator/parameter/model. Since the value of MSE

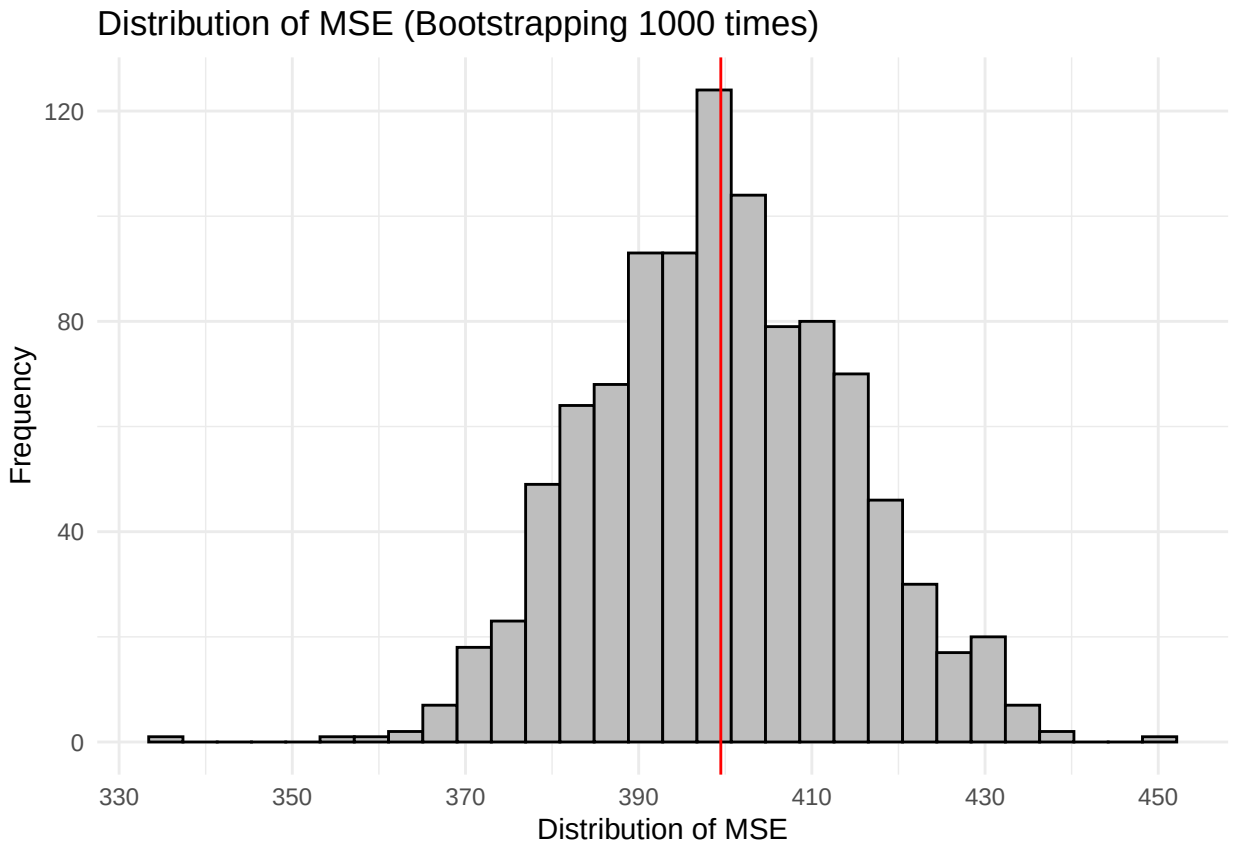
(395.2701693) we obtained in question 1 where we used the entire dataset as the training set is larger than the value of MSE we obtain in this question (389.1582007) where we split the dataset into two -train and test set- using the simple holdout validation approach, the second question where we use only the test set to calculate MSE indicates a better estimate at the data points in question. To put in other words, according to the MSE values obtained, the simple holdout approach in question 2 produces a lower average error than the model using the entire data in question 1, suggesting that the holdout approach provides us with a more accurate learner. This then indeed signals to use what we touched upon in question 1, that we may have been overfitting the data when using the entire data in question 1, suggesting that building a learner by using fewer observations could indeed provide us with a more accurate learner.

Question 3

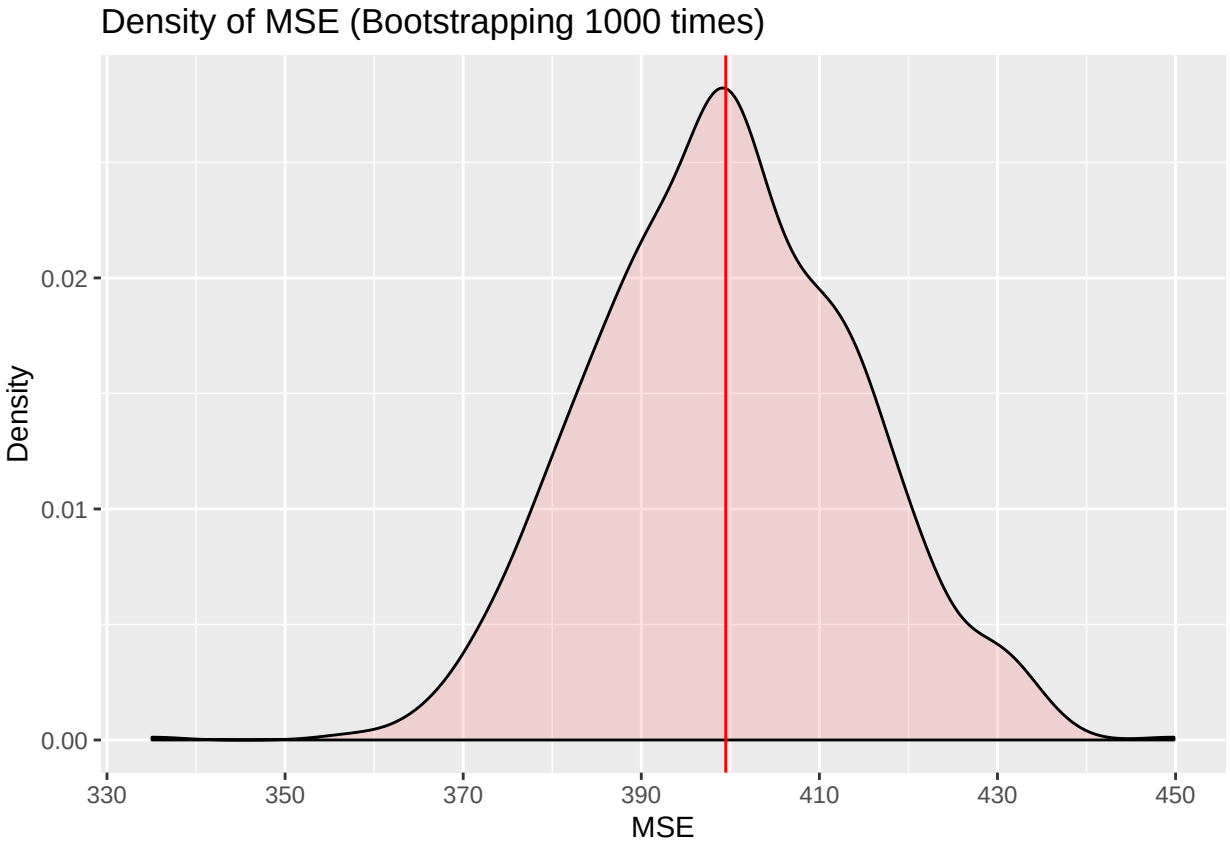
Repeat the simple validation set approach from the previous question 1000 times, using 1000 different splits of the observations into a training set and a test/validation set. Visualize your results as a sampling distribution (hint: think histogram or density plots). Comment on the results obtained.

```
set.seed(1234)
biden_boot <- replicate(1000, {
  split_biden_boot <- initial_split(data = biden_data, prop = 0.5)
  train_biden_boot <- training(split_biden_boot)
  test_biden_boot <- testing(split_biden_boot)
  training_boot <- lm(biden ~ female + age + educ + dem + rep,
                      data = train_biden_boot)
  mse_test_boot <- augment(training_boot, newdata = test_biden_boot) %>%
                     rcfss::mse(truth = biden, estimate = .fitted) %>%
  select(.estimate)%>%as.numeric()
}
)
biden_boot_df <- as.data.frame(biden_boot)

#Plotting histogram
ggplot(biden_boot_df) +
  geom_histogram(aes(biden_boot), boundary = 1, color="black", fill="gray") +
  theme_minimal() +
  geom_vline(xintercept =  mean(biden_boot_df$biden_boot), color = "red") +
  scale_x_continuous(breaks = seq(330, 450, 20)) +
  labs(x = "Distribution of MSE", y = "Frequency",
       title = "Distribution of MSE (Bootstrapping 1000 times)")
```




```
#Plotting density
ggplot(as.data.frame(biden_boot), aes(biden_boot)) +
  geom_density(alpha=.2, fill="#FF6666") +
  geom_vline(xintercept =  mean(biden_boot_df$biden_boot), color = "red") +
  scale_x_continuous(breaks = seq(330, 450, 20)) +
  labs(x = "MSE", y = "Density",
       title = "Density of MSE (Bootstrapping 1000 times)")
```



As it can be seen from the above histogram and density plots, using the simple validation approach 1000 times while doing 1000 different splits of the observations into a training set and a test/validation set, yielded MSE values that are centered around 400, or more precisely 399.4852229. Therefore, since this value is pretty close to the original MSE value we obtained (395.2701693) in question 1, we can conclude that the bootstrapped MSEs actually do converge (approximately) to the true MSE we have for the original sample (Q1).

Question 4

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.

```
set.seed(1234)
lm_coefficients <- function(splits, ...){
  mod <- lm(..., data = analysis(splits))
  tidy(mod)
}

biden_boot_1000 <- biden_data %>% as.tibble() %>%
  bootstraps(1000) %>%
  mutate(coef = map(splits, lm_coefficients,
                    as.formula(biden ~ female + age + educ + dem + rep)))
```

```
## Warning: `as.tibble()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
#Coefficients from bootstrapping
boot_coefficients <- biden_boot_1000 %>%
```

```
  unnest(coef) %>%
  group_by(term) %>%
  summarize(.estimate = mean(estimate),
                              .se = sd(estimate, na.rm = TRUE)) %>%
  rename(Boot.Estimate = .estimate, Boot.Std.Error = .se)

#Coefficients from original linear model from Q1
lm_model_coefficients <- data.frame(sum_model$coefficients) %>%
  mutate(term = names(lm_model$coefficients)) %>%
  rename(Orig.Estimate = Estimate, Orig.Std.Error = Std..Error) %>%
  select(term, Orig.Estimate, Orig.Std.Error)

#Merging coefficients from original and bootstrapping
all_coefficients <-left_join(boot_coefficients, lm_model_coefficients, by = "term")

knitr::kable(all_coefficients)
```

term	Boot.Estimate	Boot.Std.Error	Orig.Estimate	Orig.Std.Error
(Intercept)	58.9618075	2.9498903	58.8112590	3.1244366
age	0.0475608	0.0284700	0.0482589	0.0282474
dem	15.4294207	1.1110451	15.4242556	1.0680327
educ	-0.3511933	0.1920187	-0.3453348	0.1947796
female	4.0755294	0.9488085	4.1032301	0.9482286
rep	-15.8871021	1.4315343	-15.8495061	1.3113624

As we can see from the table above, the bootstrap method produces very similar values for the coefficient estimates compared with the original linear model (the differences in the estimates are only at the decimal level). Regarding the standard errors, the bootstrap method produces a slighly bigger standard errors for age, democract, female and republican variables and lower standard error for education variable; however again the difference is mostly at the hundreds (decimal). This slight difference may stem from the fact that while the original regression (from Q1) relies on assumptions regarding the underlying distribution of the data, the bootstrapping method does not (assumes unknown distribution). In fact, bootstrap method is a resampling technique for estimating a sampling distribution. Hence, when we are averse to making distibutional assumptions and/or have only access to a smaller sample rather than the entire population, bootstrapping could allows us to make more robust estimates.