Journal: Cross Validation

Zane Billings

10 February, 2020

Breakdown of Stein Method

1. Generate data (or obtain real data in correct format).
2. Partition the data into thirds (i.e. we are setting up for 3-fold cross validation).
3. Use two of the parts to build a model with a given smoothing parameter λ.
4. Use the third part to test the model, and compute the sum of square errors (SSE) for the model.
5. Repeat steps 4 and 5 for each possible testing dataset.
6. Compute the arithmetic mean of the three SSEs. This value is called the cross-validation error (CVE).
7. Repeat steps 2-6 for several values of λ.

Questions to ask before writing code

1. Is this procedure correct or am I missing something?

   This procedure is correct according to Stein's method. Dr. Wagaman also confirmed this is standard practice.

2. Why is $k = 3$ ok here? Every resource I have seen suggests $k = 5$ or $k = 10$ (or "leave-one-out" cross-validation for large datasets).

   This probably has to do with the trade-off between computational power and parameter fitting. If $k = 3$ recovers the parameters, there is not a compelling reason to do more computations.

3. Do I randomly sample the dataset each time I choose a new λ value, or do I only do a random partition at the beginning of the CV process?

   No, the data should only be partitioned at the beginning of the cross-validation process.

To-Do List:

1. Write a function to randomly partition the data.
2. Make sure the function to fit a model with a given *lambda* works as intended (I think it does but I'm going to give it a quick lookover).
3. Write a SSE function (might already be in the old stuff).
4. Write a CVE function.
5. Write a function that takes in a dataset and a value of lambda and does cross-validation.
6. Modify this function or write a wrapper to accept a list of lambdas and output a data frame of lambdas and CVEs.

Part 1: Randomly Partitioning the Data

The first thing I wanted to do was randomly generate a really simple dataframe to test how to randomly partition the data.

```
df <- data.frame(x = seq(1,100,1), y = seq(2,200,2))
```

Now, I have an idea for how to do this manually, so I will write it out for the case $k = 3$.

```
num_obs <- nrow(df)
k <- 3
num_each <- floor(num_obs/k)
# Set a random seed to ensure I can replicate results.
set.seed(100)
# Create a list of all remaining row indices.
rem <- seq(1,num_obs,1)
# Randomly sample 33 row indices.
choices <- sample(rem, 33, replace = F)
# Create the first partition, and remove all chosen indices from the list.
part1 <- df[choices, ]
rem <- setdiff(rem, choices)
# Repeat as above.
choices <- sample(rem, 33, replace = F)
part2 <- df[choices, ]
rem <- setdiff(rem, choices)
# Now set all remaining indices to the final partition.
part3 <- df[rem, ]
# Create a list of partitions so I can check for equality.
parts1 <- list(part1, part2, part3)
```

Now that I know how to do what I want, it's time to automate this process with a function.

```
random_partition <- function(data, k) {
  # data is a data frame of observations
  # k is the number of partitions to form
  # Returns a list of k random partitions of the data
  num_obs <- nrow(data)
  part_size <- floor(num_obs/k)
  rem <- seq(1, num_obs, 1)
  parts <- list()
  for (i in 1:(k - 1)) {
    choices <- sample(rem, part_size, replace = F)
    parts[[i]] <- data[choices, ]
    rem <- setdiff(rem, choices)
  }
  parts[[k]] <- data[rem, ]
  return(parts)
}
```

Here, I am testing to see if the functional output is identical to the manual result.

```
set.seed(100)
parts2 <- random_partition(df, 3)
identical(parts1,parts2)
```

```
## [1] TRUE
```

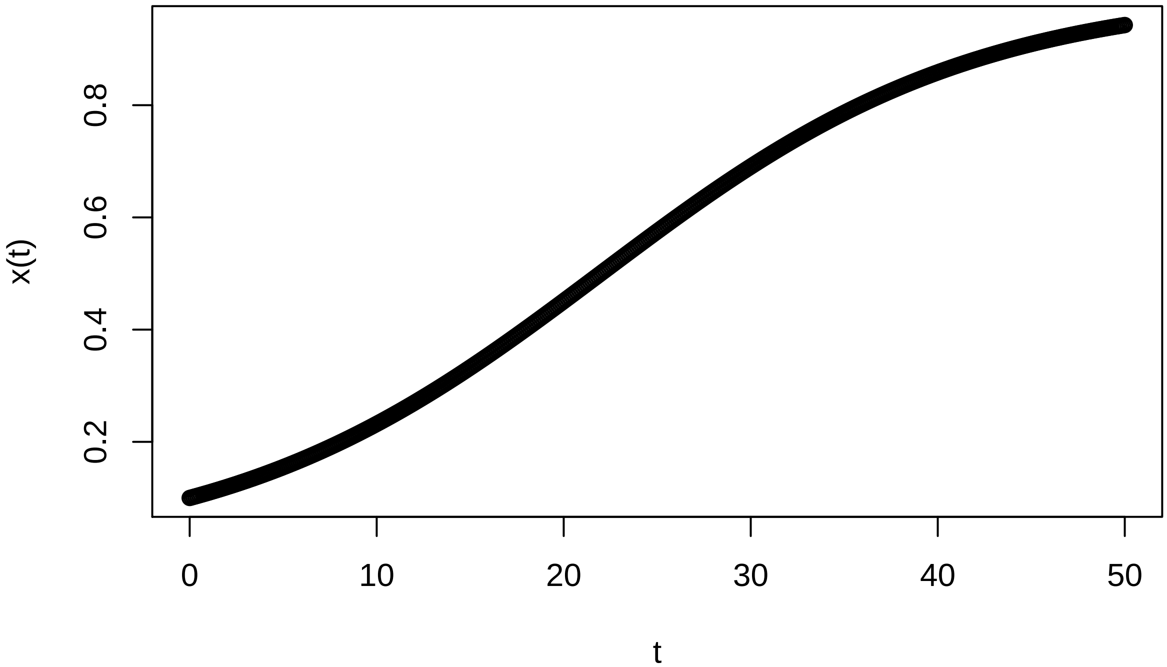
Part 2: Generating fake data and doing the thing

```r
source(here::here("Scripts","Dimensionless_exploration.R"))
```

```
## Loading required package: MASS
```

```r
source(here::here("Scripts", "Helpers.R"))

df <- generate_dimensionless_logistic_data(.1, 0.1, 50, 0.1, TRUE)
```



Now the data is generated with parameter $r = 0.1$.

```r
calculate_SSE <- function(true, fits) {
  # Calculate the sum of squared errors for a model.
  SSE <- sum((true - fits)^2)
  return(SSE)
}

lambdas <- c(0.001, 0.01, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000)
k <- 3
parts <- random_partition(df, k)
combos <- combn(1:k, k-1, simplify = F)
```

Now let's run do a test with one parameter value to see if I can do this right.

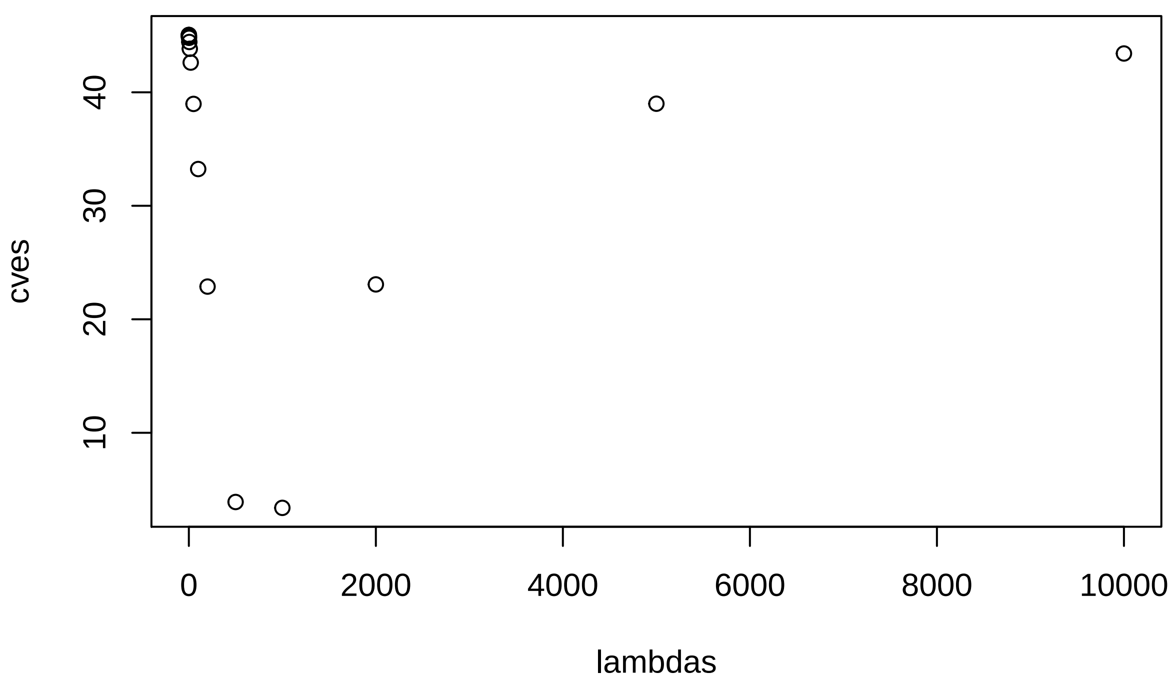
```r
source(here::here("Scripts", "Least_squares_methods.R"))
# Holder for calculated CVEs
cves <- numeric(length(lambdas))
for (a in 1:length(lambdas)) {
l = lambdas[[a]] # Grab the value of lambda
errors <- numeric(length(parts)) # Holder for SSEs
# Need to do this k number of times (# of parts)
for (i in 1:k) {
  # Grab the first combo of indices to use
```

```r
  to_use <- combos[[i]]
  # Make a blank dataframe
  train <- data.frame()
    # This part has to be done k-1 times to build training data.
  for (j in to_use) {
    # Get the next data frame.
    new_dat <- parts[[j]]
    # Add this data frame to the bottom of the rest.
    train <- rbind(train, new_dat)
  }
  # Now train has k-1 of the parts in it.
  # Set test to be the part that wasn't selected.
  not_used <- (1:k)[!(1:k %in% to_use)]
  test <- parts[[not_used]]
  # Prep and model the data, specify time step manually.
  prepped <- prep_data(train, 0.1)
  mod <- model_logistic_data_dimensionless_smoothing(prepped, l)
  # Next need to calculate predicted values using the estimated growth rate from the model.
  #P = (P0e^(rt))/(1+P0(1-e^(rt)))
  t <- test$t
  pred <- (0.1*exp(mod*t))/(1+0.1*(exp(mod*t)-1))
  errors[i] <- calculate_SSE(test$P,pred)
}

cves[a] <- mean(errors)}

this <- data.frame(lambdas,cves)
```

```r
plot(this)
```



To-Do 2020-03-24:

1. Write a line of code to grab the lambda value with the lowest CVE
2. Write a function to automate this whole process.
3. Make the results function stop automatically printing to console.
4. Run this for a bunch of different values of r and see what happens–plot r vs. min CVE lambda to see if there is a pattern.
5. Check that as lambda goes to infty, estimate of r goes to zero.
6. WOrk on 2 species.
7. Work on code organization (e.g. package?) and update manuscript.