Feature selection for SAE

Nairobi Workshop: Day 3 (afternoon)

Ann-Kristin Kreutzmann

Josh Merfeld

August 26, 2024

Feature selection

- Let's start with some example data I have
  - This comes from Malawi
    - Northern Malawi only (due to the size of the data)
  - And we'll use it all day tomorrow!

▼ Code

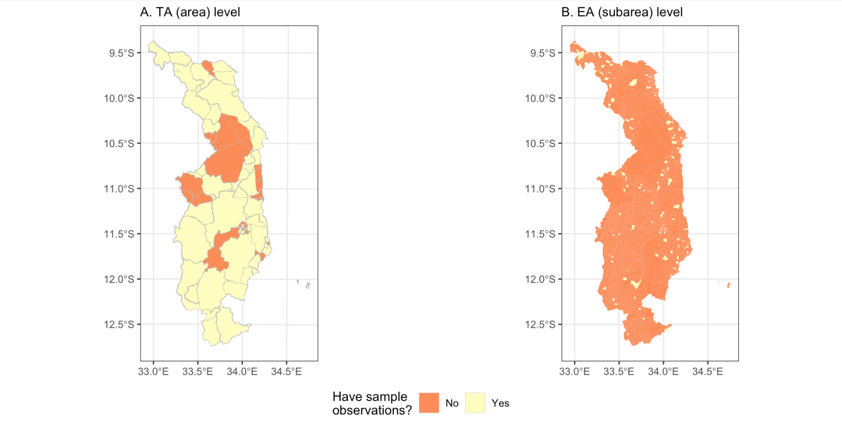
```
library(tidyverse)
surveycollapsed <- read_csv("day3data/ihs5ea.csv")
predictors <- read_csv("day3data/mosaikvars.csv")
```

A short explanation

- The survey data is collapsed to the admin3 level (TAs)
    - This is the `area`, in SAE terminology
    - I have poverty rates for `areas` (TAs) and `subareas` (EAs)
    - I have some variables that predict poverty at the `subarea` level

- So it's a perfect setup for SAE!
    - We want to estimate poverty at the TA
    - We don't have any observations in some TAs and we have too few in others
    - We could estimate a subarea model

Observations?



Predictive features

- I also have a bunch of predictive features!
  - The data come from something called MOSAIKS, that we'll discuss briefly tomorrow
  - In short, they are variables derived from satellite imagery
  - Take a look at this

▼ Code

```
predictors
```

```
# A tibble: 2,911 × 501
    EA_CODE  mosaik1  mosaik2 mosaik3 mosaik4 mosaik5 mosaik6 mosaik7 mosaik8 mosaik9 mosaik10 mosaik11 mosaik12 mosaik13 mosaik1
      <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl
 1 10101001 0.00143  0.00242    0.632  0.0334  0.0684   0.223 0.00641  0.0753   0.172    0.200    0.385  0.00697     1.51 0.00345
 2 10101002 0.000659 0.000250   0.911  0.0468  0.0978   0.357 0.00359  0.127    0.249    0.291    0.623  0.00565     1.83 0.00085
 3 10101003 0.000657 0.000403   0.811  0.0373  0.0794   0.326 0.00285  0.100    0.196    0.249    0.532  0.00426     1.77 0.00064
 4 10101004 0.00102  0.000769   0.975  0.0578  0.111    0.369 0.00584  0.140    0.264    0.316    0.666  0.00852     1.92 0.00116
 5 10101005 0.000472 0.000351   0.815  0.0344  0.0668   0.381 0.00287  0.0947   0.172    0.257    0.526  0.00484     1.75 0.00078
 6 10101006 0.00107  0.000835   0.861  0.0496  0.122    0.315 0.00536  0.137    0.255    0.281    0.644  0.00834     1.71 0.00190
```

```
 7 10101007 0.00132  0.000842    1.13   0.0549  0.0999    0.594 0.00649  0.154     0.235     0.389     0.789  0.00820     2.25 0.00165
 8 10101008 0.00202  0.00182     1.05   0.0796  0.166     0.415 0.00953  0.179     0.309     0.347     0.794  0.0116      1.94 0.00308
 9 10101009 0.000445 0.000417    0.834  0.0332  0.0663    0.375 0.00278  0.0950    0.168     0.263     0.522  0.00452     1.82 0.00068
10 10101010 0.000720 0.000438    0.794  0.0367  0.0849    0.328 0.00377  0.109     0.195     0.255     0.566  0.00556     1.63 0.00113
# ℹ 2,901 more rows
# ℹ 341 more variables: mosaik160 <dbl>, mosaik161 <dbl>, mosaik162 <dbl>, mosaik163 <dbl>, mosaik164 <dbl>, mosaik165 <dbl>, mos
#   mosaik246 <dbl>, mosaik247 <dbl>, mosaik248 <dbl>, mosaik249 <dbl>, mosaik250 <dbl>, mosaik251 <dbl>, mosaik252 <dbl>, mosaik
```

We have a problem

▼ Code

```r
# this is how many subarea observations we have
nrow(surveycollapsed)
```

```
[1] 107
```

▼ Code

```r
# this is how many predictors we have
ncol(predictors)
```

```
[1] 501
```

- What's the problem?

- It's actually impossible to estimate a model with more predictors than observations!

Another problem: overfitting

- There's another problem, too

- If we have too many predictors, we can "overfit" the model
  - This means the model is too complex
  - It fits the data we have *too* well
  - This means it doesn't generalize well to new data

- So we need to select the best predictors
  - What does "best" mean here?

Generalizing out-of-sample

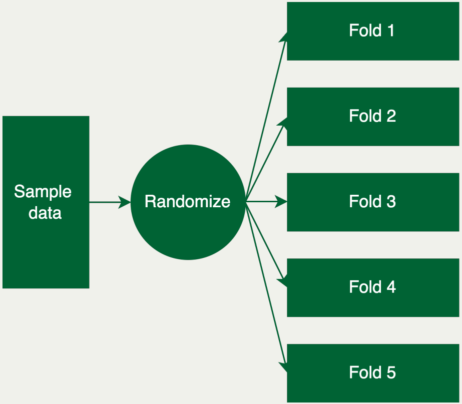
- We want to know what best predicts OUT of sample

- So we are going to set up our data to allow this:
  - We will split the data into X parts
  - A common number for X is 10, but let's do 5

Cross validation

Sample data

Cross validation



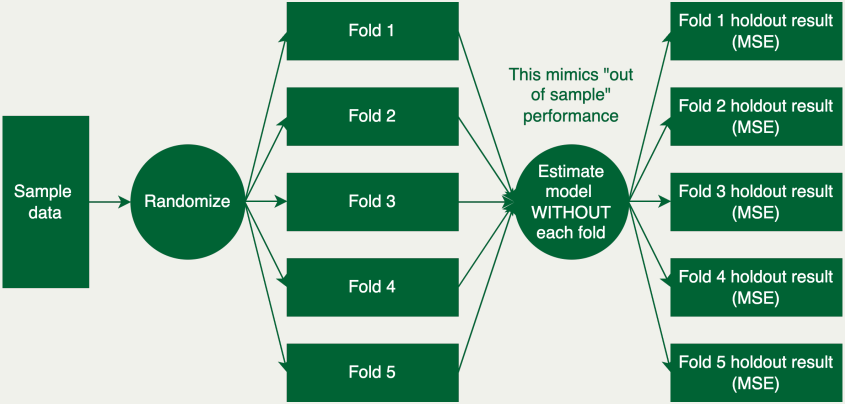
Cross validation - random folds

▼ Code

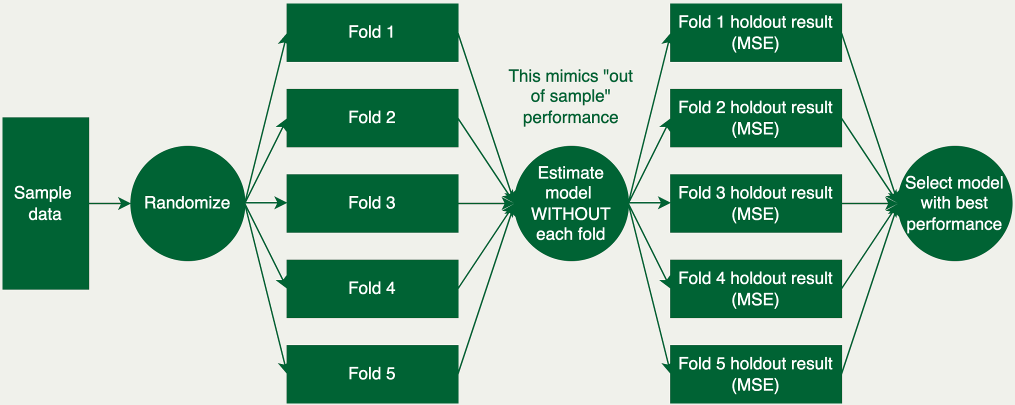
```r
surveycollapsed$fold <- sample(1:5, nrow(surveycollapsed), replace = TRUE)
head(surveycollapsed)
```

```
# A tibble: 6 × 5
   EA_CODE   poor total_weights total_obs  fold
     <dbl>  <dbl>         <dbl>     <dbl> <int>
1 10101006 0.230          5690.        16     2
2 10101011 0.444          7614.        16     2
3 10101027 0.0947         9441.        16     4
4 10101033 0.376          7486.        16     2
5 10101039 0.600          9147.        16     1
6 10101054 0.497          5351.        16     5
```

Cross validation



Cross validation



But what “models” are we going to fit?

- What are the models we are going to fit?
  - We want a way to select the best predictors
  - This will reduce the number of predictors and prevent overfitting (we hope)

- We are going to use a method called LASSO (or lasso)
  - It’s an acronym: **L**east **A**bsolute **S**hrinkage and **S**election **O**perator
  - No details, but it’s a way to select the best predictors
    - It “penalizes” the coefficients of the predictors
  - `R` package `glmnet` does this for us

The setup - with a transformed outcome

▼ Code

```r
library(glmnet)
set.seed(398465) # this is a random process, so we want to set the seed!

# we need to set up the data (combining the predictors and the outcome)
data <- surveycollapsed |>
  left_join(predictors, by = "EA_CODE")

# cv.glmnet will set up everything for us
lasso <- cv.glmnet(
  y = asin(sqrt(data$poor)), # the outcome
  x = data |> dplyr::select(starts_with("mosaik")) |> as.matrix(), # the predictors (as.matrix() is required)
  weights = data$total_weights, # the weights (sample weights)
  nfolds = 5) # number of folds (10 is the default)
lasso
```

```
Call:  cv.glmnet(x = as.matrix(dplyr::select(data, starts_with("mosaik"))),      y = asin(sqrt(data$poor)), weights = data$total_weights,
nfolds = 5)

Measure: Mean-Squared Error

     Lambda Index Measure       SE Nonzero
```

What have we done?

▼ Code

```
lasso
```

```
Call:  cv.glmnet(x = as.matrix(dplyr::select(data, starts_with("mosaik"))),      y = asin(sqrt(data$poor)), weights = data$total_weights,
nfolds = 5)

Measure: Mean-Squared Error

     Lambda Index Measure       SE Nonzero
min 0.02030    26 0.04227 0.006409       6
1se 0.06493     1 0.04418 0.005811       0
```

- What are the different “models”?
  - Different values of lambda
  - In this case, the “best” lambda is 0.02030
  - Note that some people prefer to use the `1se` value (it is more conservative). No details today.

Different values of lambda: different predictors!

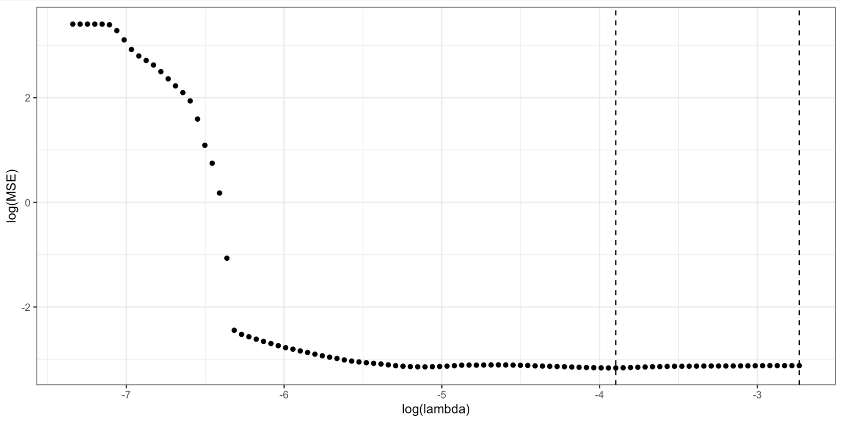
▼ Code

```
lasso
```

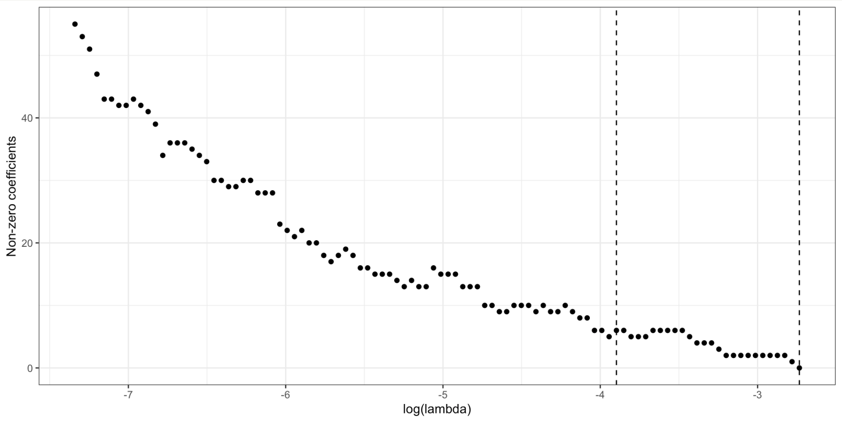
```
Call:  cv.glmnet(x = as.matrix(dplyr::select(data, starts_with("mosaik"))),      y = asin(sqrt(data$poor)), weights = data$total_weights,
nfolds = 5)

Measure: Mean-Squared Error

     Lambda Index Measure       SE Nonzero
min 0.02030    26 0.04227 0.006409       6
1se 0.06493     1 0.04418 0.005811       0
```

- At the “optimal” lambda, we have 6 predictors (non-zero coefficients)

Choosing based on mean-squared error (MSE)



Non-zero coefficients



Non-zero coefficients

▼ Code

```r
coef(lasso, s = "lambda.min")
```

```
501 x 1 sparse Matrix of class "dgCMatrix"
                       s1
(Intercept)   0.568658061
mosaik1       .
mosaik2       .
mosaik3       .
mosaik4       .
mosaik5       .
mosaik6       .
mosaik7       .
mosaik8       .
mosaik9       .
mosaik10      .
mosaik11      .
mosaik12      .
mosaik13      .
mosaik14      .
mosaik15      .
mosaik16      .
```

What we want: the non-zero variable names!

- Getting the names of the variables is more complicated than it should be

▼ Code

```r
# first, turn the coefs into a data.frame
coefs <- coef(lasso, s = "lambda.min") |>
  as.matrix() |>
  as.data.frame()
coefs
```

```
                      s1
(Intercept)  0.568658061
mosaik1      0.000000000
mosaik2      0.000000000
mosaik3      0.000000000
mosaik4      0.000000000
mosaik5      0.000000000
mosaik6      0.000000000
mosaik7      0.000000000
mosaik8      0.000000000
mosaik9      0.000000000
mosaik10     0.000000000
mosaik11     0.000000000
mosaik12     0.000000000
```

```
mosaik13      0.000000000
mosaik14      0.000000000
mosaik15      0.000000000
mosaik16      0.000000000
mosaik17      0.000000000
```

What we want: the non-zero variable names!

- Getting the names of the variables is more complicated than it should be

▼ Code

```r
# Now, create variable that is the name of the rows
coefs$variable <- rownames(coefs)
head(coefs)
```

```
                   s1    variable
(Intercept) 0.5686581 (Intercept)
mosaik1     0.0000000     mosaik1
mosaik2     0.0000000     mosaik2
mosaik3     0.0000000     mosaik3
mosaik4     0.0000000     mosaik4
mosaik5     0.0000000     mosaik5
```

▼ Code

```r
# non-zero rows
coefs <- coefs[coefs$s1!=0,]
# finally, the names of the variables
coefs$variable
```

```
[1] "(Intercept)" "mosaik39"    "mosaik234"   "mosaik277"   "mosaik280"   "mosaik396"   "mosaik459"
```

One more step: remove the Intercept!

- We don’t want the name of the intercept
    - All of the packages we use will add that automatically

▼ Code

```
allvariables <- coefs$variable[-1]
allvariables
```

```
[1] "mosaik39"  "mosaik234" "mosaik277" "mosaik280" "mosaik396" "mosaik459"
```

How do we use this with ebp?

- In EBP, we need a `formula`
- How do we turn this into a formula?
  - We need to add the outcome variable (poor) `and` combine the predictors with `+`

▼ Code

```
ebpformula <- as.formula(paste("poor ~", paste(allvariables, collapse = " + ")))
ebpformula
```

```
poor ~ mosaik39 + mosaik234 + mosaik277 + mosaik280 + mosaik396 +
    mosaik459
```

Finally: estimating the model

▼ Code

```r
library(povmap) # I like to use povmap instead of emdi (personal preference)
# get "area" variable
data$TA_CODE <- substr(data$EA_CODE, 1, 5)
data$TA_CODE <- substr(data$EA_CODE, 1, 5)
ebp <- ebp(fixed = ebpformula, # the formula
  pop_data = predictors, # the population data
  pop_domains = "EA_CODE", # the domain (area) name in the population data
  smp_data = data, # the sample data
  smp_domains = "EA_CODE", # the domain (area) name in the sample data
  transformation = "arcsin", # I'm going to use the arcsin transformation
  weights = "total_weights", # sample weights
  weights_type = "nlme", # weights type
  MSE = TRUE, # variance? yes please
  L = 0) # this is a new thing in povmap: "analytical" variance estimates. much faster!
```

```
Time difference of 3.53 secs
```

▼ Code

```r
head(ebp$ind)
```

```
    Domain      Mean Head_Count
1 10101001 0.4791939 0.05260923
2 10101002 0.3958995 0.12526969
3 10101003 0.3788624 0.14668910
4 10101004 0.3884519 0.13432329
5 10101005 0.3860476 0.13734796
6 10101006 0.3256817 0.16367811
```