

LONDON  
SCHOOL of  
HYGIENE  
& TROPICAL  
MEDICINE



# Super Learner

David Whitney\*

Machine Learning 2022



\*based on Prof Karla Diaz-Ordaz's 2021 materials

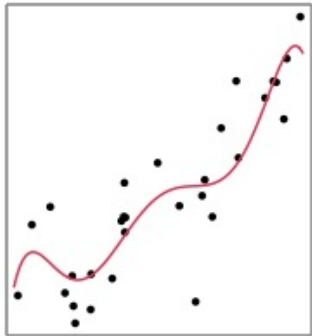
# Learning Outcomes

After completing today's session, you will

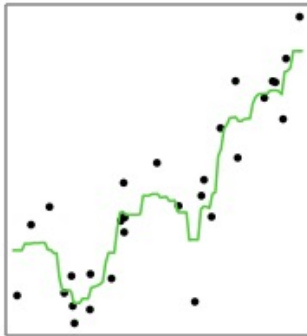
- understand the basic idea of Super Learner (SL)
- understand the 3 important choices to be made for using SL
- be able to apply SL in a practical example
- know how to use cross-validation to measure the performance of a SL

# Motivation

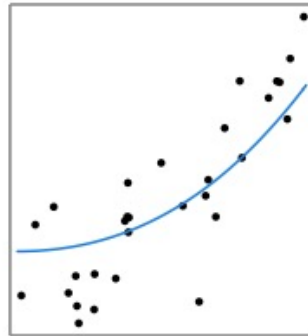
- You have experience with a wide library of prediction models (learners):



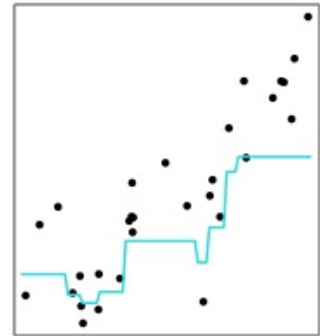
Linear Regression



Random Forest



LASSO Regression



Gradient Boosting

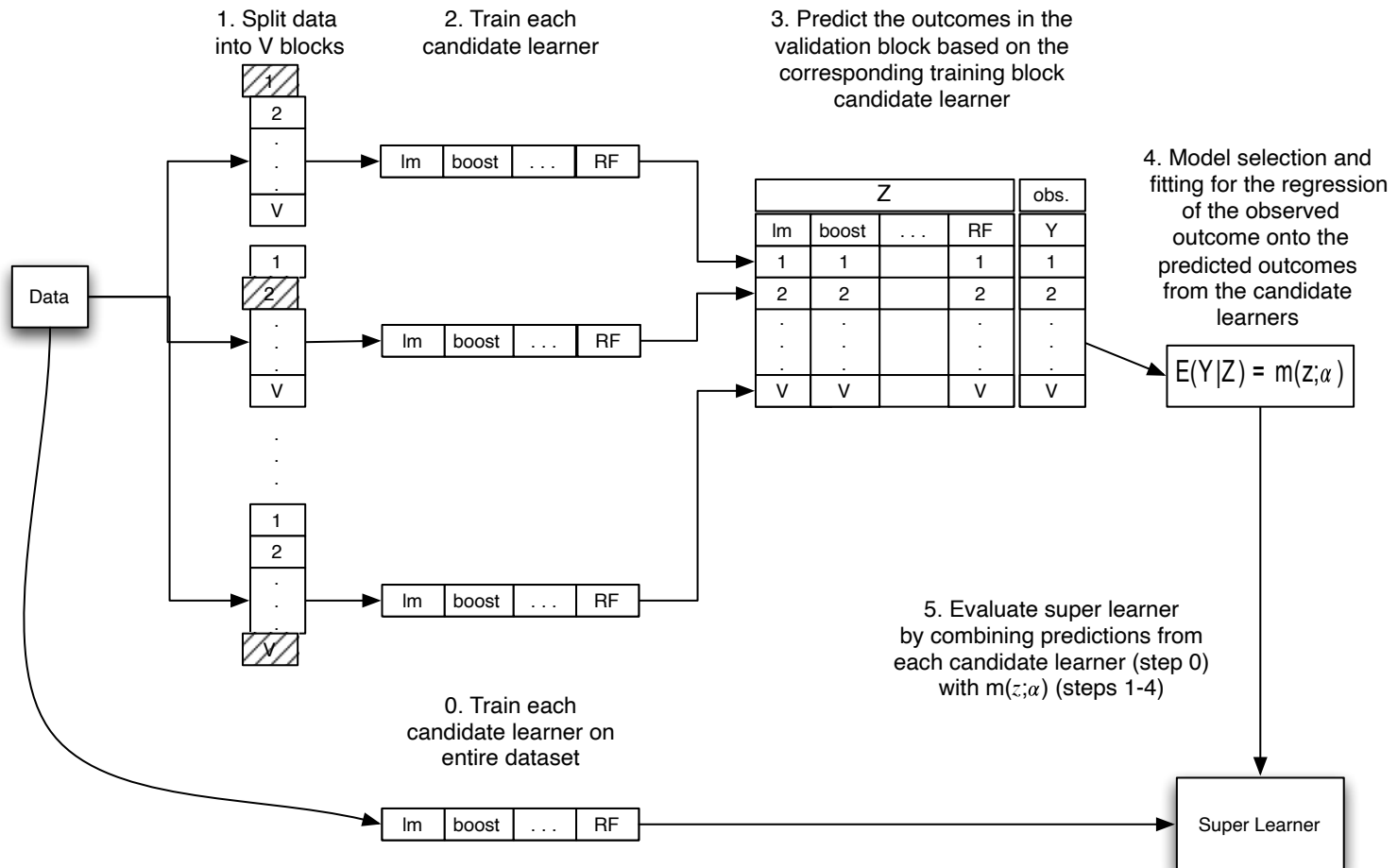
- Your collaborators/colleagues may have many more!
- For a given application, there may be several candidate learners
- How might you choose between the candidates?

# Super Learner: a bird's eye view

The **Super Learner** is a meta-learning algorithm that

- Takes the data and a library  $\mathcal{L}$  of  $L$  candidate learners as inputs
  - Estimates performance of each learner using  $V$ -fold cross-validation
  - Creates an optimal ensemble  $m(Z; \alpha)$  of the learners
- 
- The ensemble  $m(Z; \alpha)$  offers a reproducible way to choose from (or combine) predictions from the candidate learners
  - Super Learner ensembles also possess desirable (large sample) guarantees

# Super Learner: a bird's eye view



# What makes a Super Learner?

Before we can follow the steps from above, we must specify

1. The library  $\mathcal{L}$  of candidate learners
2. A loss function  $\ell$  to measure performance
3. Form of the meta-learner  $m(Z; \alpha)$

# Library

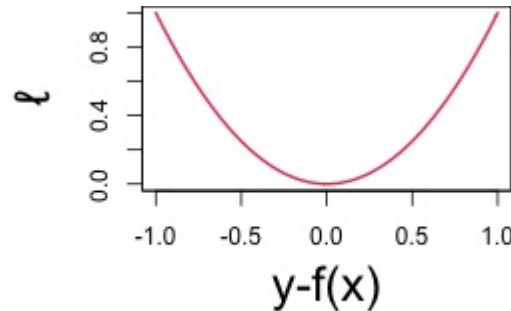
- Common algorithms to consider

- Linear regression
- Polynomial linear regression
- Random forest
- Bagging
- GAMs
- Gradient boosting
- Neural network
- Polynomial spline regression

- Select the learner algorithms using contextual knowledge of the problem
- If a custom-made algorithm (eg a clinical prediction model) is known to perform well, include it!
- It is recommended to include a parametric model

# Loss function

- The loss function is often the squared error loss  $\ell = (Y - f(X))^2$



- For binary outcomes where  $Y = 0, 1$ :
  - Negative binomial log likelihood
  - Rank loss function: maximises the area under the ROC curve (a function of both sensitivity and specificity), thus optimizing the algorithms ability to correctly classify observations.



# Meta-regression model

- We often use non-negative least squares (NNLS)

$$E[Y|X = x] = \alpha_1 \hat{f}_1(x) + \dots + \alpha_L \hat{f}_L(x)$$

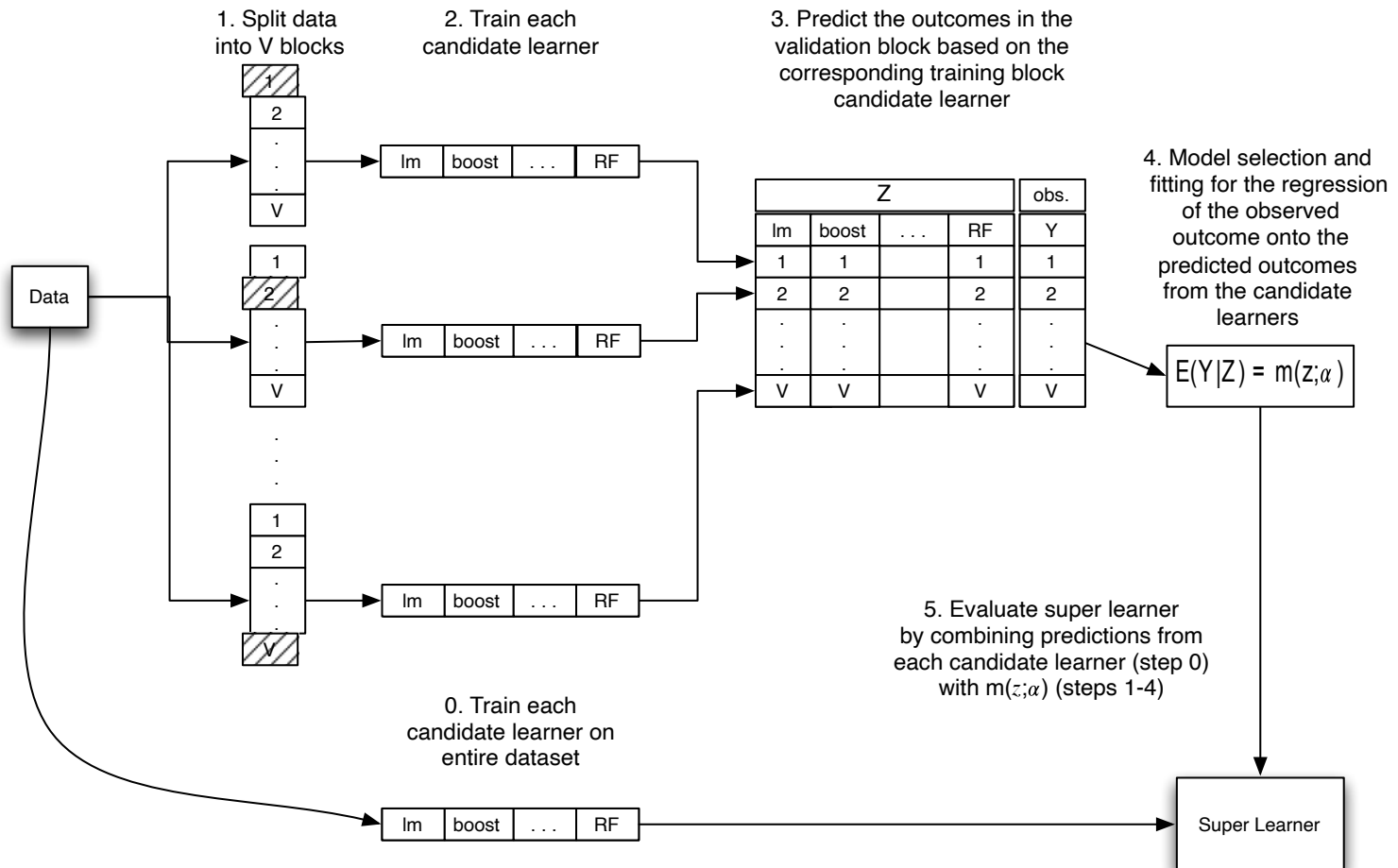
- In NNLS,  $\alpha_l \geq 0$  and  $\sum_{l=1}^L \alpha_l = 1$
- Thus  $m(Z_i; \alpha)$  is a convex combination of the learners in  $\mathcal{L}$  with weights  $\alpha$  chosen to minimise the CV risk of SL:

$$\hat{\alpha} = \arg \min_{\alpha} \sum_i (Y_i - m(Z_i; \alpha))^2$$

- We obtain a **discrete super learner** by instead placing all the weight on the learner  $\hat{f}_l$  that performs best using  $V$ -fold CV

# Super Learner in R

# Super Learner: a bird's eye **view**



# Super Learner in R

- You will need the **SuperLearner** package installed for the practical:

```
SL_installed <- require(SuperLearner)
if(!SL_installed) install.packages("SuperLearner")
library(SuperLearner)
```

# The SuperLearner function

- The main function in **SuperLearner** is itself called **SuperLearner**:

```
args(SuperLearner)
```

```
## function (Y, X, newX = NULL, family = gaussian(), SL.library,  
##      method = "method.NNLS", id = NULL, verbose = FALSE, control = list(),  
##      cvControl = list(), obsWeights = NULL, env = parent.frame())  
## NULL
```

Key arguments for customising your Super Learner:

- **SL.library**: character vector of learners (this is  $\mathcal{L}$ )
- **method**: method to estimate meta-learner (based on  $\ell$  and  $m(Z|\alpha)$ )
- Enter **?SuperLearner** in your console for more details

# Note: when coding learners for your $\mathcal{L}$ ...

- **SuperLearner** uses a standard interface to automate training/prediction
- This is achieved via wrapper functions that follow this template:

```
SuperLearner::SL.template
```

```
## function (Y, X, newX, family, obsWeights, id, ...)  
## {  
##   if (family$family == "gaussian") {  
##     }  
##   if (family$family == "binomial") {  
##     }  
##   pred <- numeric()  
##   fit <- vector("list", length = 0)  
##   class(fit) <- c("SL.template")  
##   out <- list(pred = pred, fit = fit)  
##   return(out)  
## }  
## <bytecode: 0x7f8e92cda5c8>  
## <environment: namespace:SuperLearner>
```

# Built-in wrappers

- You **might** be able to use a built-in wrapper for your learner

```
listWrappers("SL")
```

```
## All prediction algorithm wrappers in SuperLearner:
```

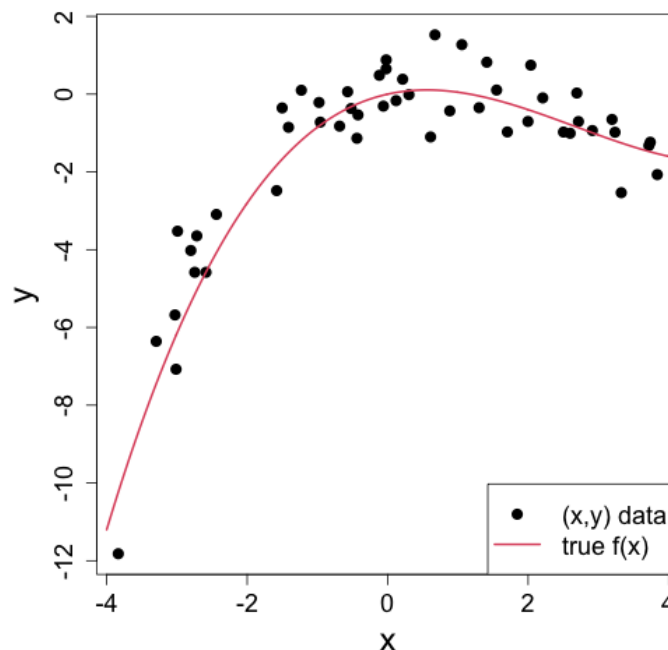
```
## [1] "SL.bartMachine"      "SL.bayesglm"
## [3] "SL.biglasso"         "SL.caret"
## [5] "SL.caret.rpart"     "SL.cforest"
## [7] "SL.earth"           "SL.extraTrees"
## [9] "SL.gam"              "SL.gbm"
## [11] "SL.glm"              "SL.glm.interaction"
## [13] "SL.glmnet"           "SL.ipredbagg"
## [15] "SL.kernelKnn"        "SL.knn"
## [17] "SL.ksvm"             "SL.lda"
## [19] "SL.leekasso"         "SL.lm"
## [21] "SL.loess"            "SL.logreg"
## [23] "SL.mean"             "SL.nnet"
## [25] "SL.nnls"             "SL.polymars"
## [27] "SL.qda"              "SL.randomForest"
## [29] "SL.ranger"           "SL.ridge"
## [31] "SL.rpart"            "SL.rpartPrune"
```

# Example: a univariate Super Learner

Generating some data:

```
set.seed(2490) #reproducible
n <- 50
x <- runif(n, min=-4, max=4)
f <- function(x)
{
  .4*x - .4*x^2 + .05*x^3
}
y <- f(x) + rnorm(n)
```

The data and (unknown) regression:





- Let's use GLM and random forest with the NNLS meta-learner

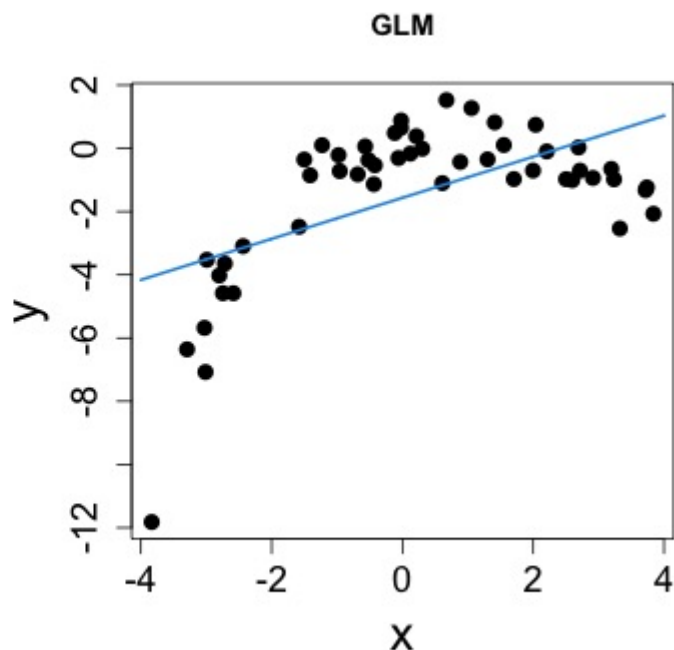
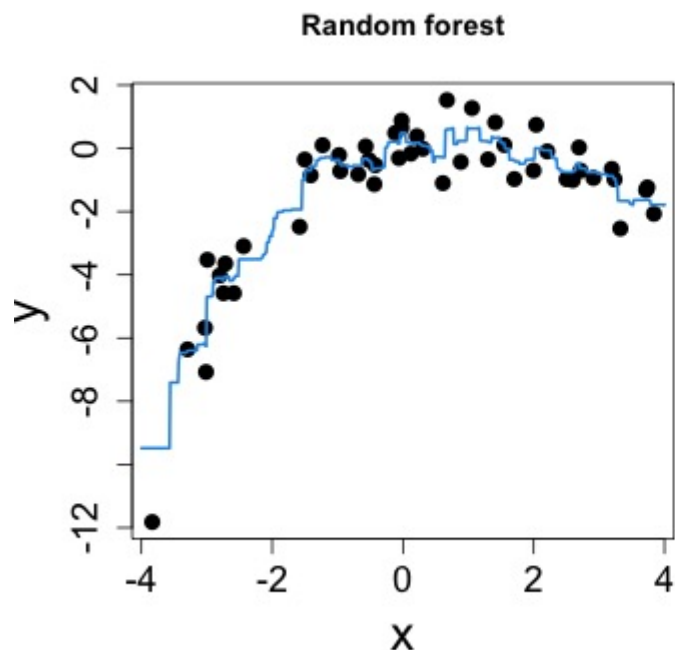
```
my.lib <- c("SL.randomForest", "SL.glm")

fit <- SuperLearner(Y = y, X = data.frame(x),
                  newX = data.frame(x=x0),
                  SL.library = my.lib,
                  method = "method.NNLS",
                  cvControl = list(V=5L),
                  control = list(saveCVFitLibrary=TRUE))

SL.pred <- fit$SL.predict
```

# Step 0. fit each learner in $\mathcal{L}$ on all data

The  $L \times n$  matrix of full-data predictions is stored as `fit$library.predict`



# Step 1. Split the data into $V = 5$ blocks

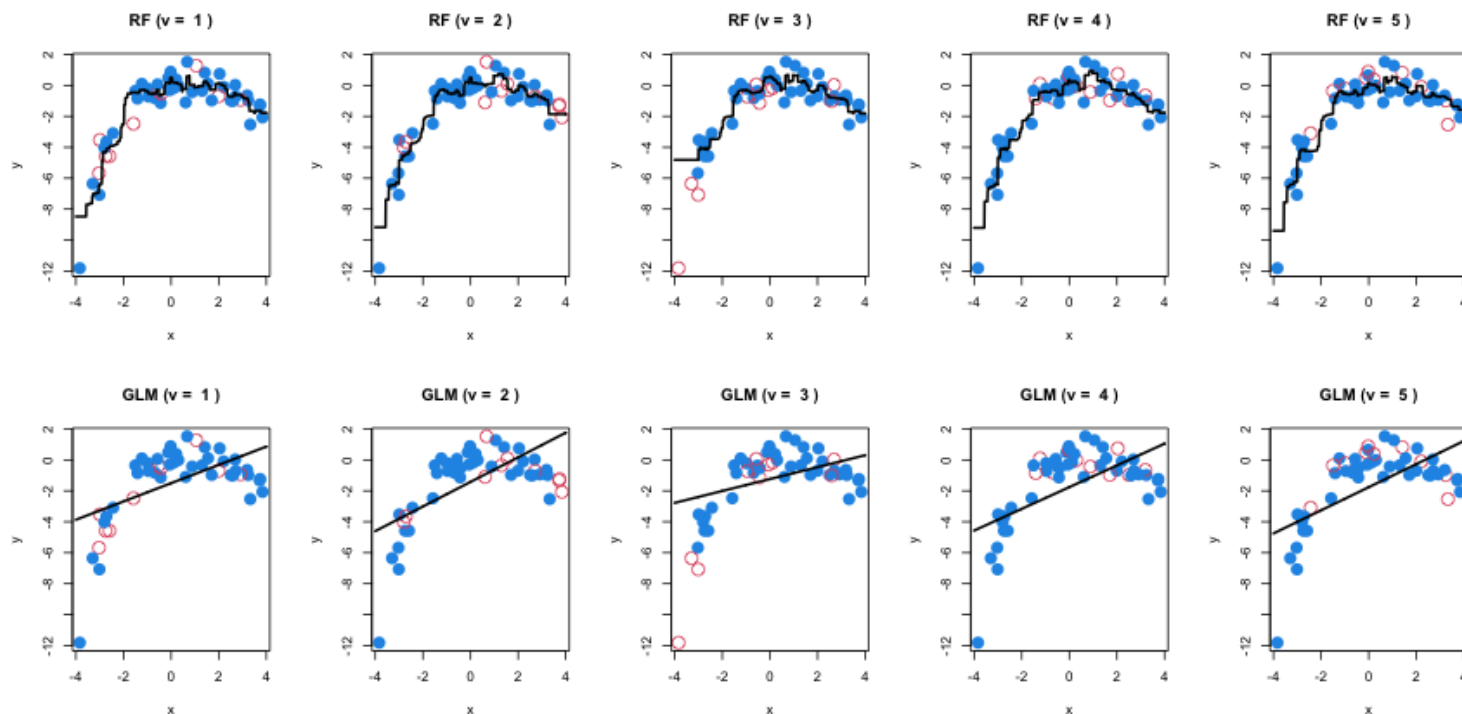
The row numbers for the  $V$ -fold cross-validation are stored as a list

```
fit$validRows #think "validation sets 1-5"
```

```
## $`1`  
## [1] 31 35 44 20 22 13 42 25 11 18  
##  
## $`2`  
## [1] 39 28 40 15 24 4 45 49 37 6  
##  
## $`3`  
## [1] 21 26 47 7 36 43 9 27 14 29  
##  
## $`4`  
## [1] 19 30 48 1 50 3 34 12 10 32  
##  
## $`5`  
## [1] 8 46 17 2 5 38 33 23 41 16
```

# Steps 2-3. Train each learner, then predict

Recall that we do this  $L \times V$  times (train on  $V - 1$  folds, predict on last fold)



(We plot training sets as solid blue, test sets as empty red)

# Step 3. Fit the meta-learner

- The  $n \times L$  matrix of fitted values on the training data are stored in `fit$Z`
  - Remember each data point belongs to **exactly one** validation set!

```
head(fit$Z)
```

```
##           [,1]      [,2]
## [1,] -0.31717312 -2.2417058
## [2,] -0.80226697  0.6553930
## [3,] -1.24058623  0.4831672
## [4,] -1.86230006  1.5361682
## [5,] -0.03448410 -0.1135900
## [6,] -0.06711427 -0.9365990
```

- The coefficient vector from NNLS is available as `fit$coef`

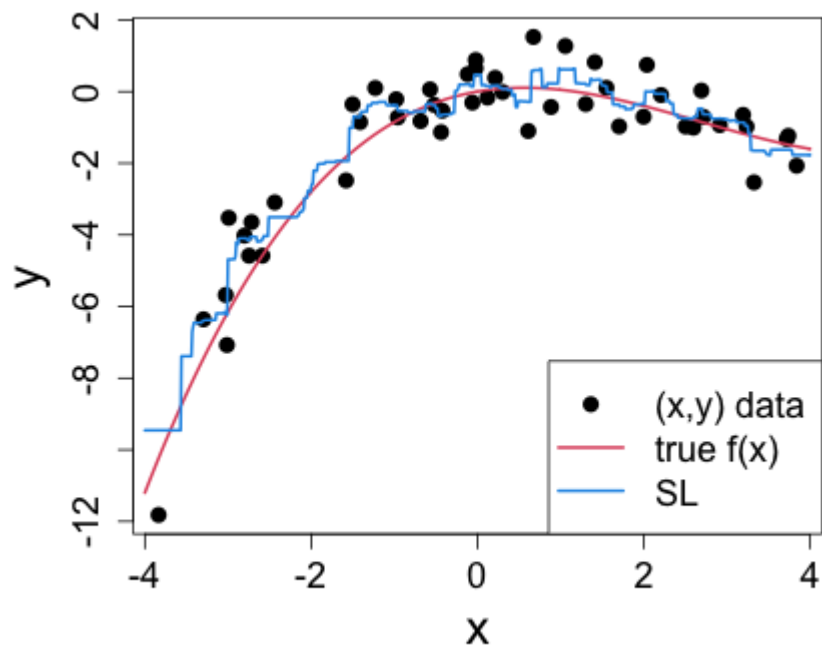
```
fit$coef
```

```
## SL.randomForest_All      SL.glm_All
##      0.994654219      0.005345781
```

- Here, most of the weight is placed on random forest

# Step 4. Predict on new data

- Predictions for `newX` are at `fit$SL.predict`



**Your turn to practice!**

... Super Learner will return in part 2