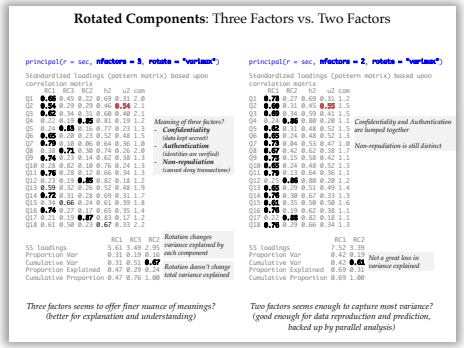
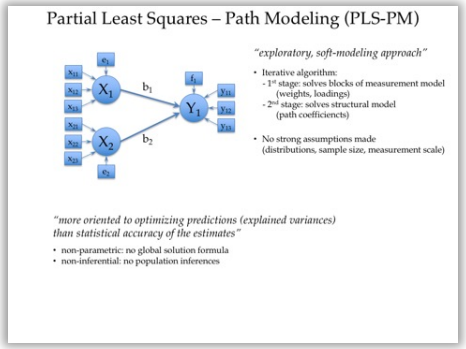
Business Analytics Using Computational Statistics

Predictions → **Ensemble Methods** → Prediction Validation & Conclusions

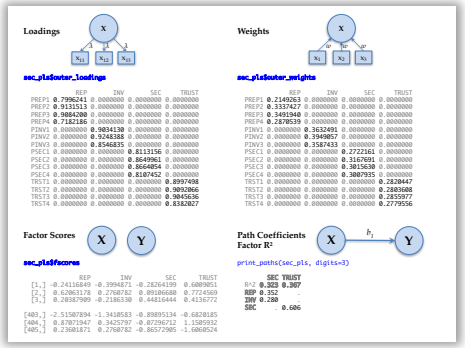
PCA Revisited



Structural Equation Modeling

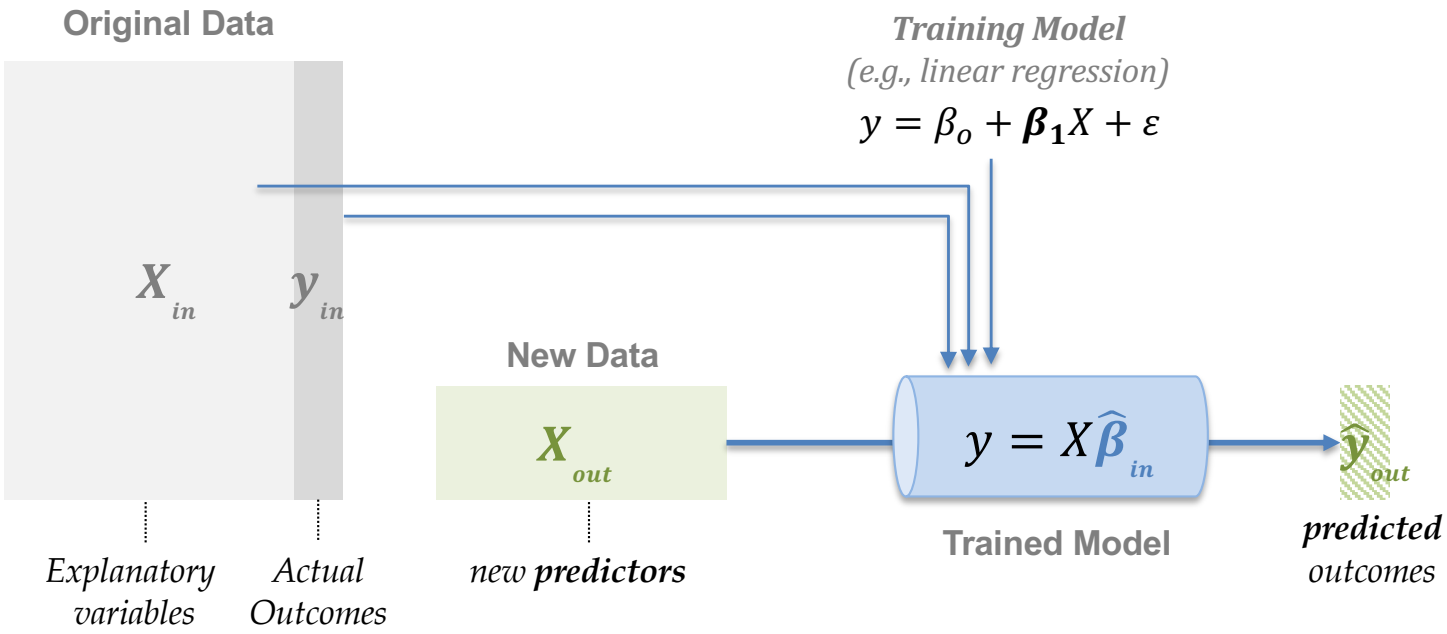


SEMinR



Generating Predictions

*We **predict** outcomes for **new data***
*using a model **trained** on original data*



Training
Model Estimation
using Original Data

Predicting
Finding Outcomes
using New Data

Our new data usually only contains predictors
We want to know the future outcomes!

MSE and Prediction

Formulas -> Models

```
cars_full           <- mpg ~ cylinders + displacement + horsepower + weight + acceleration + model_year + factor(origin)
cars_reduced        <- mpg ~ weight + acceleration + model_year + factor(origin)
cars_full_poly2     <- mpg ~ poly(cylinders, 2) + poly(displacement, 2) + poly(horsepower, 2) + poly(weight, 2) +
                             poly(acceleration, 2) + model_year + factor(origin)
cars_reduced_poly2  <- mpg ~ poly(weight, 2) + poly(acceleration,2) + model_year + factor(origin)
cars_reduced_poly6  <- mpg ~ poly(weight, 6) + poly(acceleration,6) + model_year + factor(origin)
```

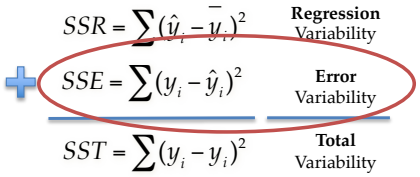
```
models = list(
  "lm_full"           = lm(cars_full, data=cars),
  "lm_reduced"        = lm(cars_reduced, data=cars),
  "lm_poly2_full"     = lm(cars_full_poly2, data=cars),
  "lm_poly2_reduced"  = lm(cars_reduced_poly2, data=cars),
  "lm_poly6_reduced"  = lm(cars_reduced_poly6, data=cars),
  "rt_full"           = rpart(cars_full, data=cars),
  "rt_reduced"        = rpart(cars_reduced, data=cars)
)
```

*Using a **named list** allows us to iterate over over all our models using `sapply` / `lapply` / etc.*

MSE and Prediction

We have seen MSE-type metrics before…

Regression Sum of Squares



MSE describes dispersion of fitting/prediction error

MSE has units²

$$MSE_{in} = \frac{\sum(y - \hat{y}_{in})^2}{n}$$

$$MSE_{out} = \frac{\sum(y_{out} - \hat{y}_{out})^2}{n}$$

```
mse <- function(errs) mean(errs^2)
mse_in <- function(model) mse(residuals(model))
mse_out <- function(actual, predicted) mse(actual - predicted)


models_mse_in <- sapply(models, mse_in)
```

	All Terms	*Reduced Terms*	
Linear Terms Regression	`lm_full` 10.682122	`lm_reduced` **10.971643**	
Polynomial Terms Regression	`lm_poly2_full` 7.919030	`lm_poly2_reduced` 8.364546	`lm_poly6_reduced` 8.254377
Regression Trees	`rt_full` 9.155146	`rt_reduced` 9.501344	

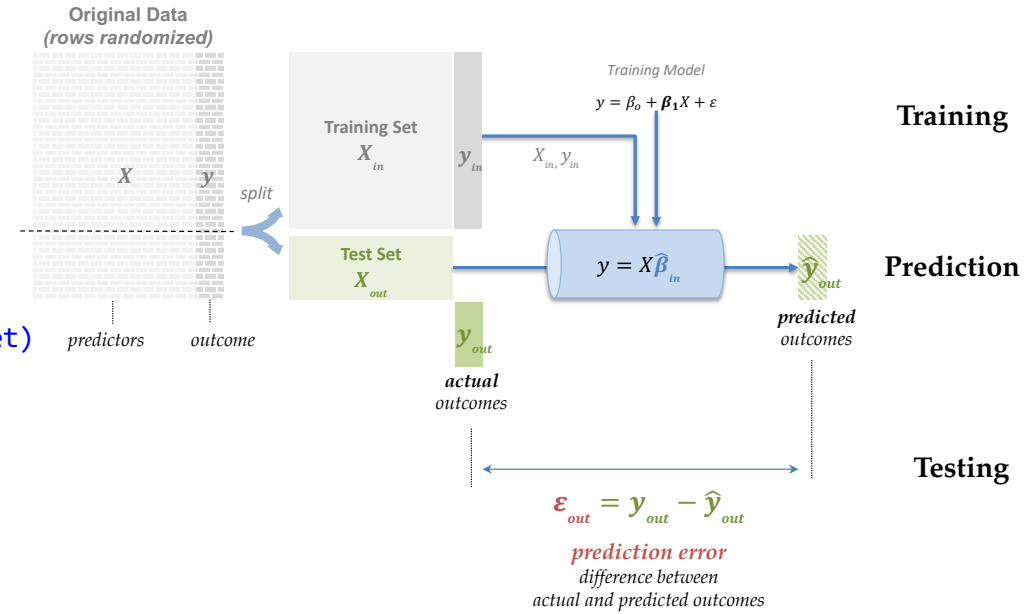
*Removing collinear terms (**multicollinearity**) does not improve model fit!*
(then why do we care about multicollinearity??)

*Using more **complex models** improves model fit*
(then why didn't we study these before??)

Split-sample Testing Revisited

Split

```
set.seed(27935752)
train_indices <- sample(1:nrow(cars), size=0.70*nrow(cars))
train_set <- cars[train_indices,]
test_set <- cars[-train_indices,]
```



Train

```
trained_model <- update(models[["lm_reduced"]], data=train_set)
```

```
mse_in(trained_model)
[1] 10.93126
```

$$MSE_{in} = \frac{\sum(y - \hat{y}_{in})^2}{n}$$

Predict

```
mpg_predicted <- predict(trained_model, test_set)
```

Test

```
mpg_actual_out <- test_set$mpg
pred_err <- mpg_actual_out - mpg_predicted
head(data.frame(mpg_actual_out, mpg_predicted, pred_err))
```

```
    mpg_actual_out mpg_predicted  pred_err
248           39.4      31.59557  7.804433
37            19.0      16.75076  2.249241
201           18.0      19.35238 -1.352376
103           26.0      28.13508 -2.135077
389           26.0      29.28920 -3.289201
100           18.0      20.39581 -2.395813
```

Out-of-sample MSE $MSE_{out} = \frac{\sum(y_{out} - \hat{y}_{out})^2}{n}$

```
mse_out(mpg_actual_out, mpg_predicted)
```

```
[1] 11.37791
```

Our MSE_{in} was **10.971643**

Out-of-sample ***prediction error is worse*** *than in-sample* ***fitting error***

Implementing k-Fold

```
fold_i_pe <- function(i, k, model, dataset, outcome) {
  folds <- cut(1:nrow(dataset), breaks=k, labels=FALSE)

  test_indices <- which(folds==i)
  test_set  <- dataset[test_indices, ]
  train_set <- dataset[-test_indices, ]
  trained_model <- update(model, data = train_set)

  predictions <- predict(trained_model, test_set)
  dataset[test_indices, outcome] - predictions
}

k_fold_mse <- function(model, dataset, outcome, k=nrow(dataset)) {
  shuffled_indicies <- sample(1:nrow(dataset))
  dataset <- dataset[shuffled_indicies,]

  fold_pred_errors <- sapply(1:k, \(kth) {
    fold_i_pe(kth, k, model, dataset, outcome)
  })

  pred_errors <- unlist(fold_pred_errors)

  mse(pred_errors)
}
```

Updating a model

*Re-estimates (**retrains**) a model with given changes*

```
update(model, data = train_set)
update(model, formula = mpg ~ displacement)
```

```
k_fold_mse(models[["lm_full"]], cars, "mpg", k=10)
[1] 11.48231

k_fold_mse(models[["lm_full"]], cars, "mpg", k=392)
[1] 11.29344
```

*More **complex models** can sometimes give us worse predictions!*

10-fold vs 392-fold:

```
models_10fold <-
  sapply(models, \(m) k_fold_mse(m, cars, "mpg", k=10))

models_loocv <-
  sapply(models, \(m) k_fold_mse(m, cars, "mpg", k=392))

in_out_errs <- data.frame(
  "mse_in" = models_mse_in,
  "10fold_cv" = models_10fold,
  "loocv" = models_loocv)

print(in_out_errs, digits = 3)
```

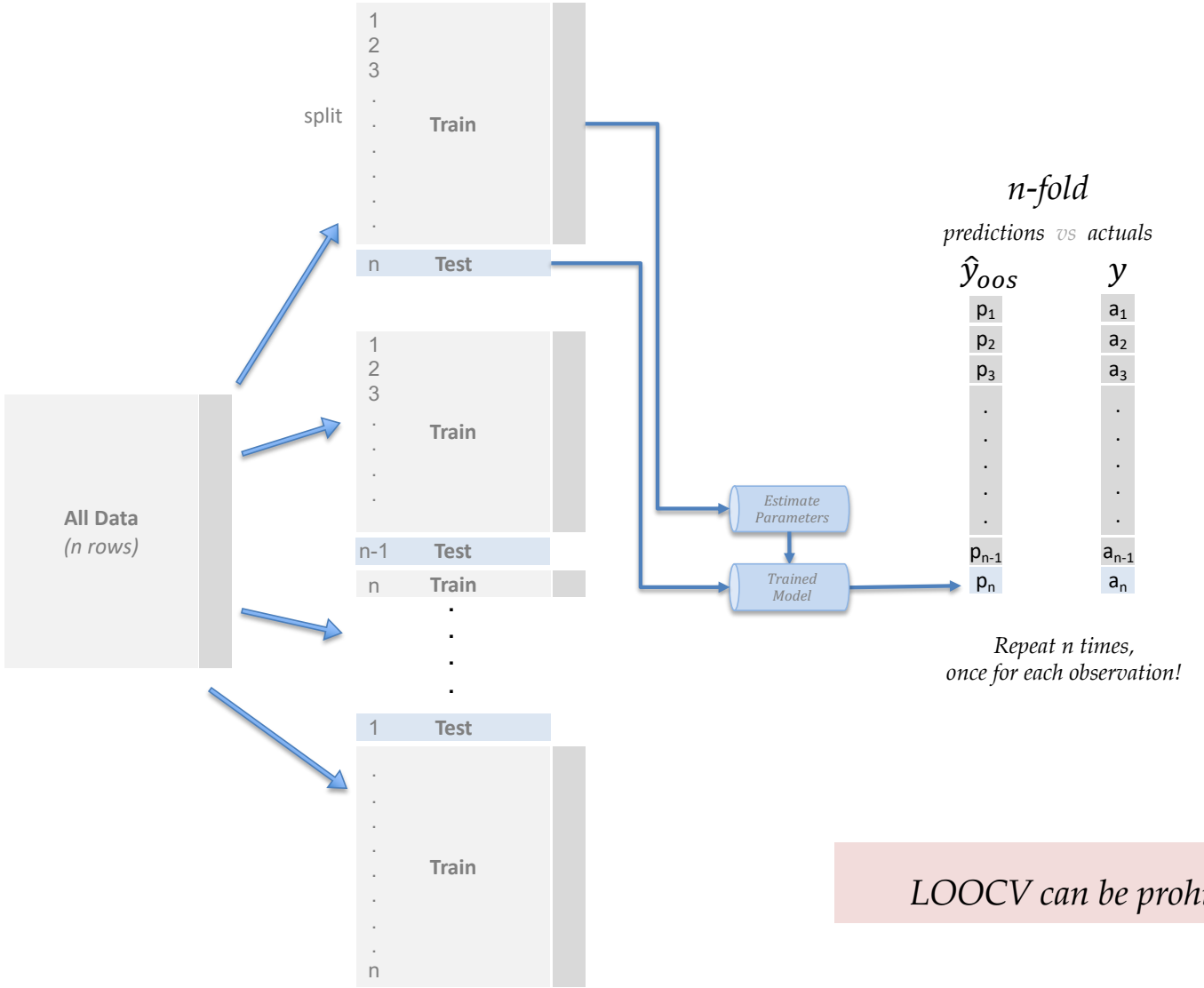
	mse_in	X10fold_cv	loocv
lm_full	10.68	11.19	11.29
lm_reduced	10.97	11.40	11.38
lm_poly2_full	7.92	8.62	8.61
lm_poly2_reduced	8.36	8.72	8.79
lm_poly6_reduced	8.25	9.19	9.18
rt_full	9.16	12.72	12.77
rt_reduced	9.50	11.60	13.15

2nd order to 6th order

Leave-One-Out Cross-Validation (LOOCV)

```
k_fold_mse(cars_log_lm, cars_log, "log.mpg.", k=392)
```

```
        is         oos
0.01332245 0.01379209
```

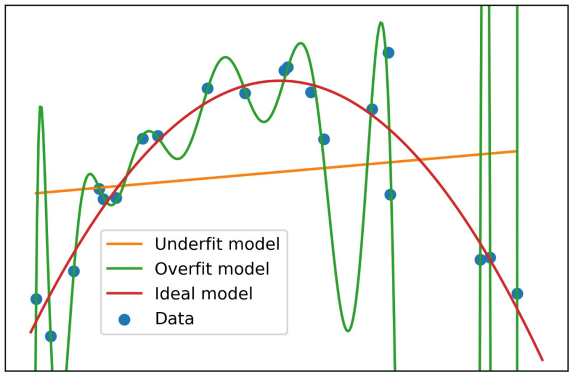


LOOCV is the least-biased k-Fold performance measure

LOOCV can be prohibitively slow for big datasets

Bias-Variance Tradeoff

https://towardsdatascience.com/overfitting-underfitting-and-the-bias-variance-tradeoff-83b42fb11efb



Better in-sample fit
does not give us
Better out-of-sample predictions

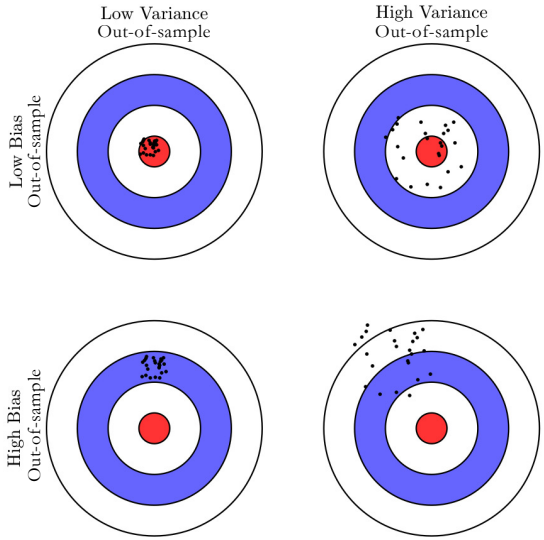
Prediction Error MSE_{out} *: total error in predicting outcome (y) of test set, given model (f) trained on training set (D)*

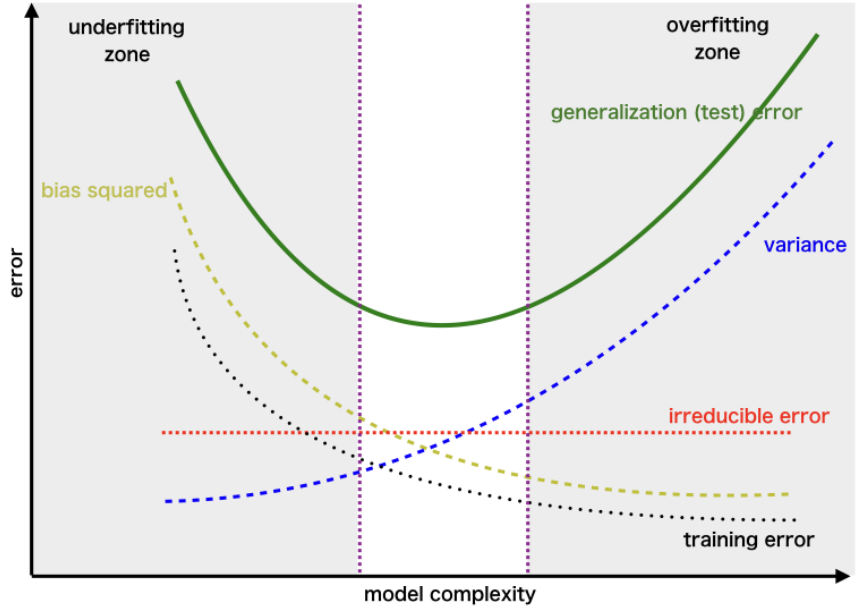
Bias*: error from missing relevant features in model (f)*

$$\mathrm{E}_{D,\varepsilon}\left[\left(y-\hat{f}\left(x;D\right)\right)^2\right]=\left(\mathrm{Bias}_D\left[\hat{f}\left(x;D\right)\right]\right)^2+\mathrm{Var}_D\left[\hat{f}\left(x;D\right)\right]+\sigma^2$$

Variance*: error from overfitting to noise in the training data (D)*

Irreducible error*: noise in relationship between DV and IVs that cannot be modeled*





source: https://mineetha.com/2020/06/25/bias-variance-tradeoff/

Ensemble Methods

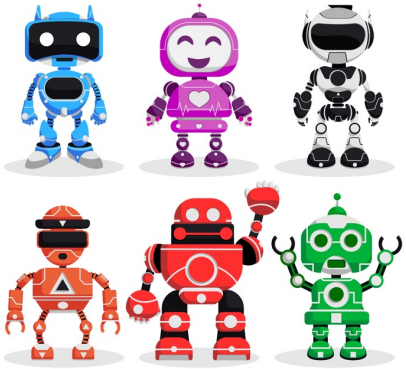
Group of separate things that contribute to a whole

When we have an important decision to make,
*we often consult many **different people**: friends, family, coworkers*

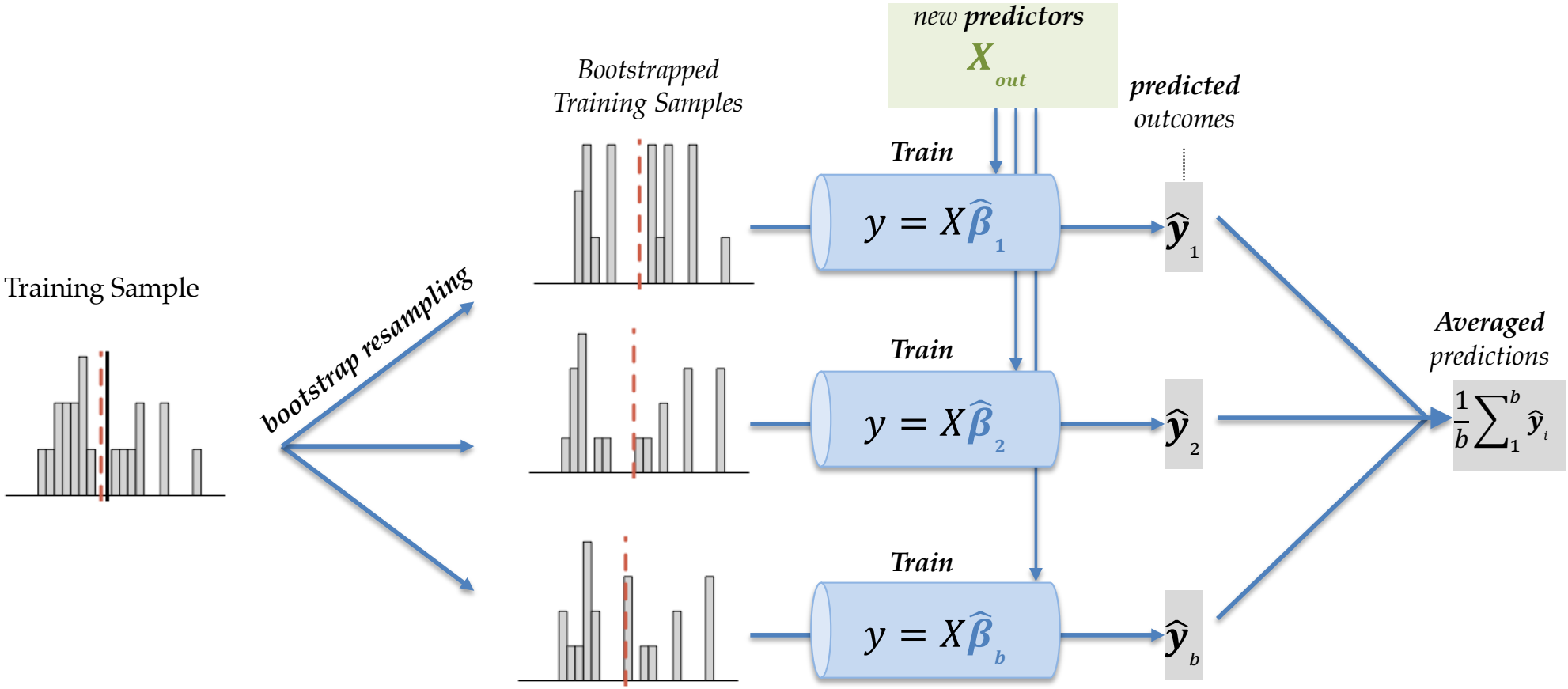
"wisdom of the crowd"

Collective opinion of a diverse independent group of individuals
rather than that of a single expert

Could machine learning be more powerful
*if we trained many **different models**?*



Bagging: **B**ootstrapped **Agg**regation




```r
bagged_learn <- function(model, dataset, b=100) {
  lapply(1:b, \(i) {
    # 1. Get a bootstrapped (resampled w/ replacement) dataset
    # 2. Return a retrained (updated) model
  })
}

bagged_predict <- function(bagged_models, new_data) {
  predictions <- lapply(…) # get b predictions of new_data
  as.data.frame(predictions) |> apply(…) # apply a mean over the columns of predictions
}
```

🤔 *Will the **b** trained models be **different enough?***

Stable Algorithms: OLS Regression

```
old_cars <- subset(cars, model_year <= 81)
new_cars <- subset(cars, model_year == 82)

mse_oos <- function(actuals, preds) {
  mean( (actuals - preds)^2 )
}
```

$$MSE_{out} = \frac{\sum(y_{out} - \hat{y}_{out})^2}{n}$$

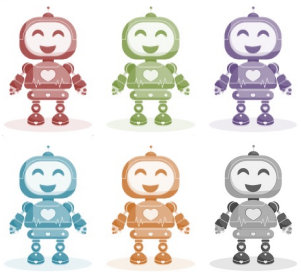
```
cars_lm <- lm(mpg ~ weight + acceleration + model_year + factor(origin), data=cars)
```

Ordinary Regression Prediction Error

```
lm_mse_out <- update(cars_lm, data = old_cars) |>
  predict(object = _, new_cars) |>
  mse_oos(new_cars$mpg, preds = _)

# [1] 15.47389
```

Bagged Regression Prediction Error

```
bagged_learn(cars_lm, old_cars, b=100) |>
  bagged_predict(bagged_models = _, new_data = new_cars) |>
  mse_oos(new_cars$mpg, preds = _)

[1] 15.51973
```

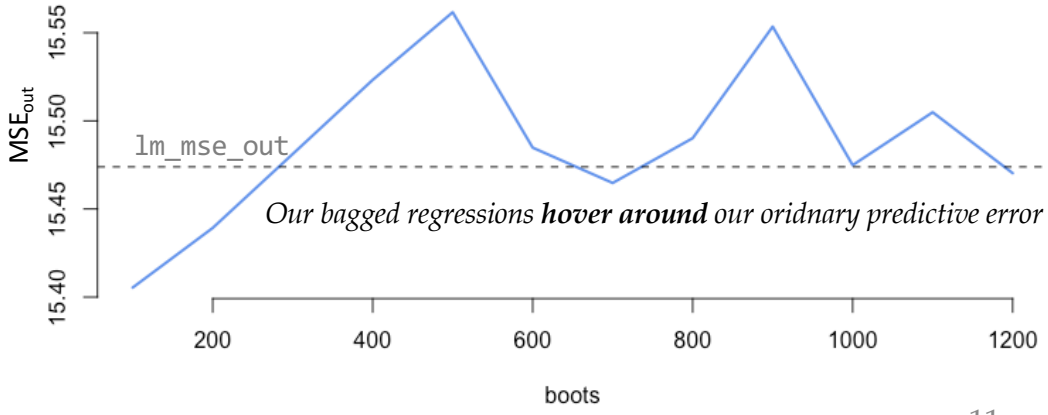


*Regression models are quite **stable**:*
*Slight changes in data produces **similar models***

Stable Algorithms with Bagging

Let's use different number of bootstraps… (100 – 1200)

```
boots <- seq(100, 1200, by = 100)

learning <- sapply(boots, \(b) {
  bagged_learn(cars_lm, old_cars, b=b) |>
    bagged_predict(new_cars) |>
    mse_oos(new_cars$mpg, preds = _)
})

plot(boots, learning, type="l")
abline(h=lm_mse_out, lty="dashed")
```



*Our bagged regressions **hover around** our oridnary predictive error*

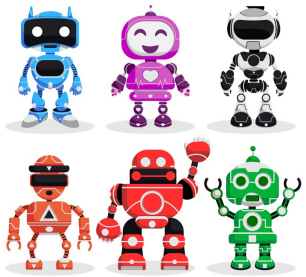
Unstable Algorithms: Decision Tree

Decision Tree Prediction Error

```
library(rpart)

old_tree <- rpart(mpg ~ cylinders + displacement + horsepower + weight +
                    acceleration + factor(origin), data=old_cars)

tree_mse_out <-
  predict(old_tree, new_cars) |>
  mse_oos(new_cars$mpg, preds = _)
```

```
# [1] 26.54635
```

Our MSE_{out} is high because trees overfit to small samples – they need tuning and more data!



Bagged Tree Prediction

```
bagged_learn(old_tree, old_cars, b=100) |>
  bagged_predict(new_cars) |>
  mse_oos(new_cars$mpg, preds = _)
```

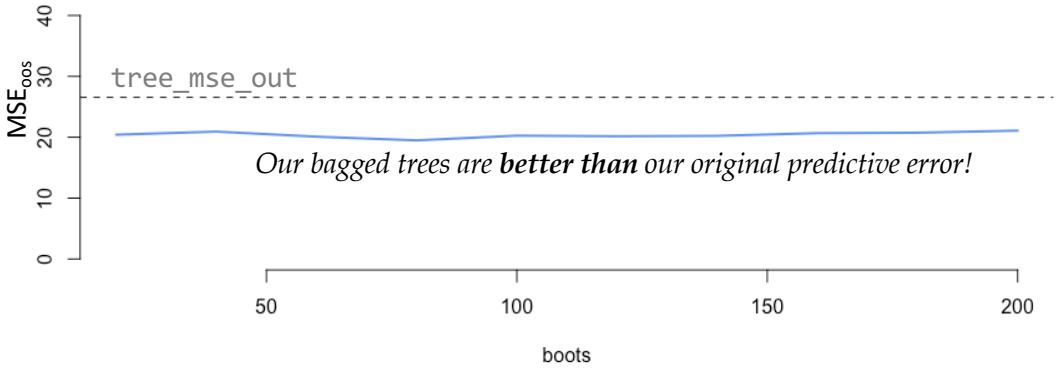
```
[1] 20.88632
```

*Decision trees are **unstable**:*
*Slight changes in data produces very **different models***

Unstable Algorithms with Bagging

Let's use different number of bootstraps…

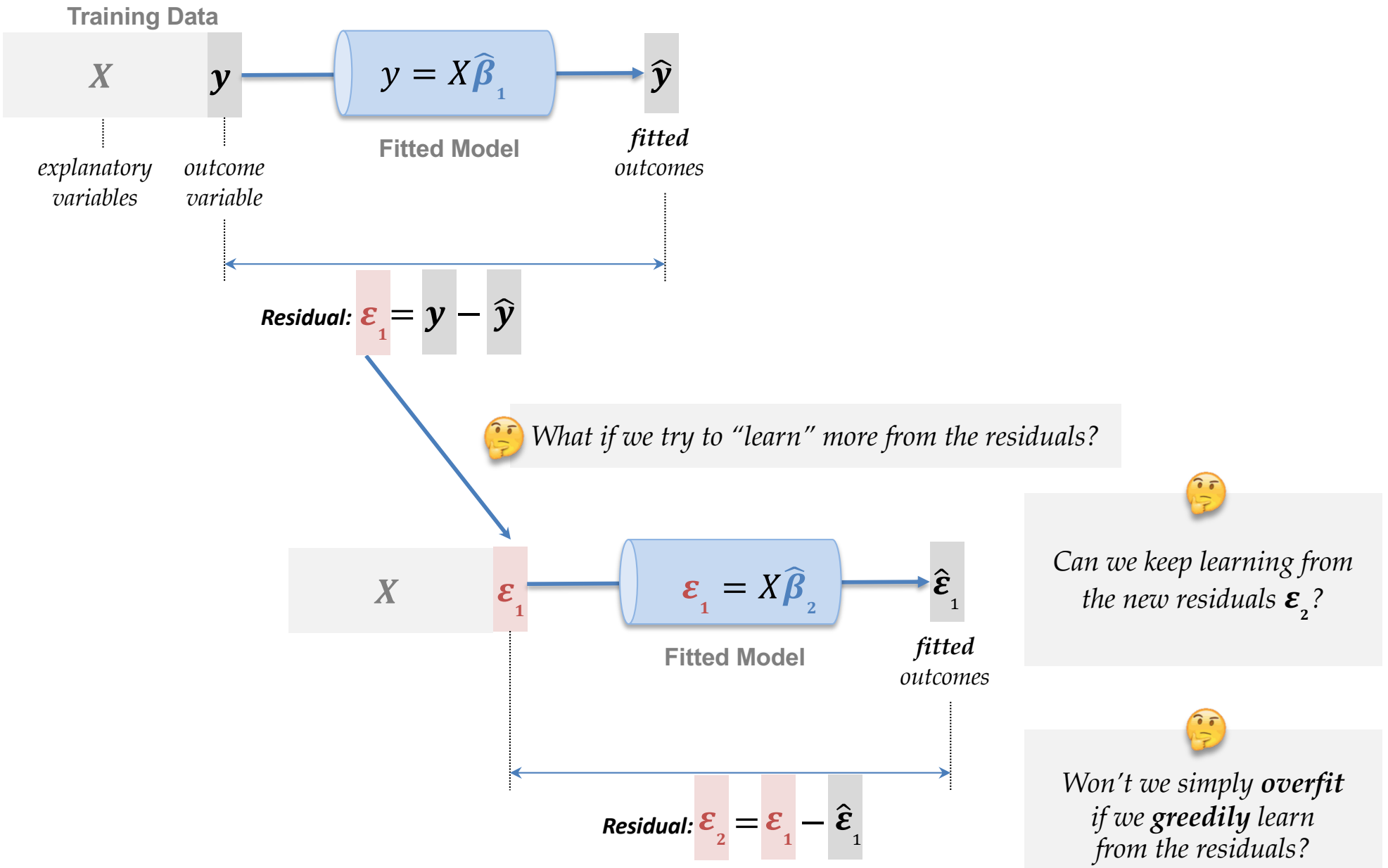
```
boots <- seq(100, 1200, by = 100)

learning <- sapply(boots, \(b) {
  bagged_learn(cars_lm, old_cars, b=b) |>
    bagged_predict(new_cars) |>
    mse_oos(new_cars$mpg, preds = _)
})

plot(boots, learning, type="l")
abline(h=old_learning, lty="dashed")
```



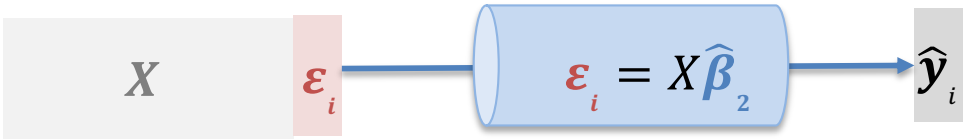
*Our bagged trees are **better than** our original predictive error!*

*Bagging decision trees is one part of a powerful predictive model called **Random Forest***

Boosting


Training Data
X
y
$y = X\hat{\beta}_1$
$\hat{y}$
Fitted Model
fitted outcomes
explanatory variables
outcome variable
Residual: $\varepsilon_1 = y - \hat{y}$
What if we try to "learn" more from the residuals?
X
ε_1
$\varepsilon_1 = X\hat{\beta}_2$
$\hat{\varepsilon}_1$
Fitted Model
fitted outcomes
Residual: $\varepsilon_2 = \varepsilon_1 - \hat{\varepsilon}_1$
Can we keep learning from the new residuals ε_2?
Won't we simply **overfit** if we **greedily** learn from the residuals?

Boosted Learning



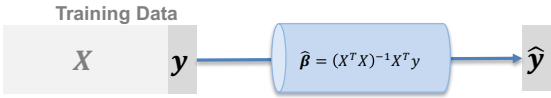
Algorithm:

Given old data with predictors and outcome: X, y

1. Start by setting the "residuals" variable $\varepsilon = y$

$$\varepsilon_0 = y$$

2. Iterate through n rounds ($i = 1..n$) training a new model f each time:
   a. Fit $f_i(X, \varepsilon)$
   b. Get fitted values $\hat{y}$ from f_i
   c. Update the residuals with *learning rate* α: $\varepsilon = \varepsilon - \alpha\hat{y}$
   d. Store each trained model f_i for predicting later



$$\varepsilon_i = \varepsilon_{i-1} - \alpha\,\hat{y}_i$$

"shrinkage" factor to slow down our learning

Result: collection of n stored models f_i

Code Skeleton:

```
boost_learn <- function(model, dataset, outcome, n=100, rate=0.1) {
  predictors <- dataset[, …] # get data frame of only predictor variables

  # Initialize residuals and models
  res <- dataset[, …] # set res to vector of actuals (y) to start
  models <- list()

  for (i in 1:n) {
    this_model <- update(model, data = cbind(mpg=res, predictors))

    res <- …          # update residuals with learning rate

    models[[i]] <- this_model # Store model
  }

  list(models=models, rate=rate)
}
```

For Loops

I generally discourage using For loops because they are lengthier version of apply functions.

***However**, for-loops are needed when we must **look back at earlier results** while iterating:*

```
result <- c(1)
for (i in 2:5) {
  result <- c(result, result[i-1] * 2)
}
```

```
[1]  1  2  4  8 16
```

Boosted Predictions

Algorithm:

Given new out-of-sample data with predictors: X_{oos}

1. Iterate through the n stored models f_i:
   a. Predict outcome for model: $\hat{y}_i = f_i(X_{oos})$
   b. Store predictions $\hat{y}_i$

2. Sum predictions together with learning rate
   a. Multiply prediction by *learning rate* α: $\boldsymbol{\alpha} f_i(X_{oos})$
   b. Sum the weighted prediction of all rounds: $E(X_{oos}) = \sum_{i=1}^{v} \boldsymbol{\alpha} f_i(X_{oos})$

Result: vector of predictions $\hat{y}_{oos}$

Code Skeleton:

```
boost_predict <- function(boosted_learning, new_data) {
  boosted_models <- …
  rate <- …
  n <- nrow(new_data)

  predictions <- lapply( … ) # get predictions of new_data from each model

  pred_frame <- as.data.frame(predictions) |> unname()

  apply( … ) # apply a sum over the columns of predictions, weighted by learning rate
}
```

Strong Learners: *OLS Regression*

$$mpg = \beta_0 + \beta_1 cyl + \beta_2 disp + \beta_3 hp + \beta_4 wt + \beta_5 acc + \beta_6 year + \beta_7 origin + \varepsilon$$

Ordinary Regression Prediction

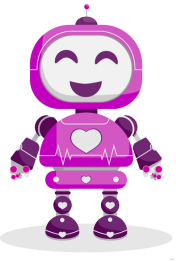
```
old_lm <- lm(mpg ~ cylinders + displacement + horsepower + weight +
               acceleration + model_year + factor(origin), data=old_cars)

mse_oos(new_cars$mpg, predict(old_lm, new_cars))
```

```
[1] 14.26774
```

Boosted Regression Prediction

```
boost_learn(cars_lm, old_cars, outcome="mpg", n=1000) |>
  boost_predict(new_cars) |>
  mse_oos(new_cars$mpg, preds = _)
```

```
[1] 15.47389
```



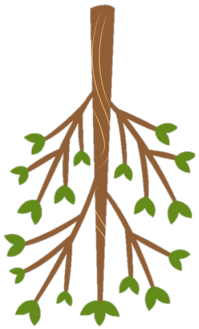
"strong learner"

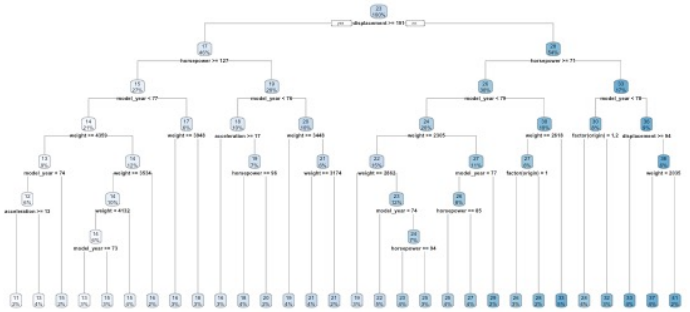
Tries to come close to an accurate answer

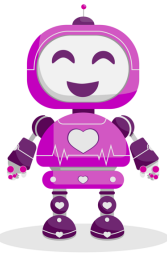
Strong learners *are not good for boosting*
They are ***too greedy*** *to learn from noise (****overfitting****)*

Strong learners are ***too similar*** *to each other*

Strong Learner: *Decision Tree*





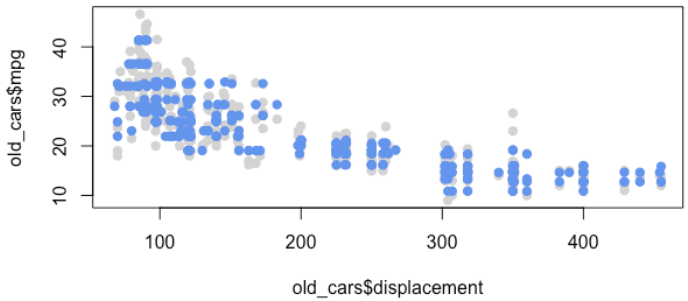


"strong learner"

```
full_tree <- rpart(mpg ~ cylinders + displacement + horsepower + weight +
                           acceleration + model_year + factor(origin),
                   data=old_cars, cp=0)

rpart.plot(full_tree)
```

Grow the tree down more fully

```
plot(old_cars$displacement, old_cars$mpg, pch=19, col="lightgray")
points(old_cars$displacement, predict(full_tree, old_cars), …)
```



A full grown tree can mimic nearly each data point

Ordinary Tree Prediction

```
mse_oos(new_cars$mpg, predict(full_tree, new_cars))
```

```
[1] 22.62033
```

Boosted Tree Prediction

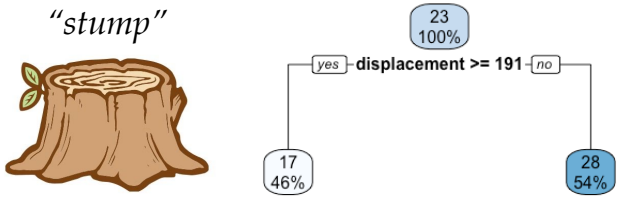
```
boost_learn(old_tree_stump, old_cars, outcome="mpg", n=1000, rate=0.01) |>
  boost_predict(new_cars) |>
  mse_oos(new_cars$mpg, preds = _)
```

```
[1] 20.35556
```

Moderate drop in predictive error

We do not see a huge improvement from boosting a full grown regression tree

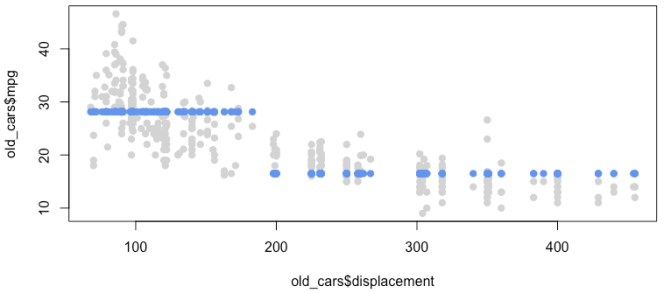
Weak Learner: *Decision Stump*



A decision tree with only a root and two leaves

```
old_tree_stump <- rpart(mpg ~ cylinders + displacement + horsepower + weight +
                        acceleration + model_year + factor(origin),
                        data=old_cars, cp=0, maxdepth=1)

rpart.plot(tree_stump)

plot(old_cars$displacement, old_cars$mpg, pch=19, col="lightgray")
points(old_cars$displacement, predict(tree_stump, old_cars), …)
```

Grow tree only to 1st level



A stump can only bin observations into two groups, based on one criteria

Ordinary Tree Stump Prediction

```
mse_oos(new_cars$mpg, predict(tree_stump, new_cars))
```

```
[1] 53.30506
```

Boosted Tree Stump Prediction

```
boost_learn(tree_stump, old_cars, outcome="mpg", n=1000, rate=0.01) |>
  boost_predict(boosted_learning = _, new_data = new_cars) |>
  mse_oos(new_cars$mpg, preds = _)
```

```
[1] 16.68617
```

Big drop in predictive error!

"weak learner"

A stump is only slightly better than random guessing

***Weak learners** improve a lot from boosting!*
*Each weak learner is **different from the others***