Optimization

Machine Learning with R

Basel R Bootcamp

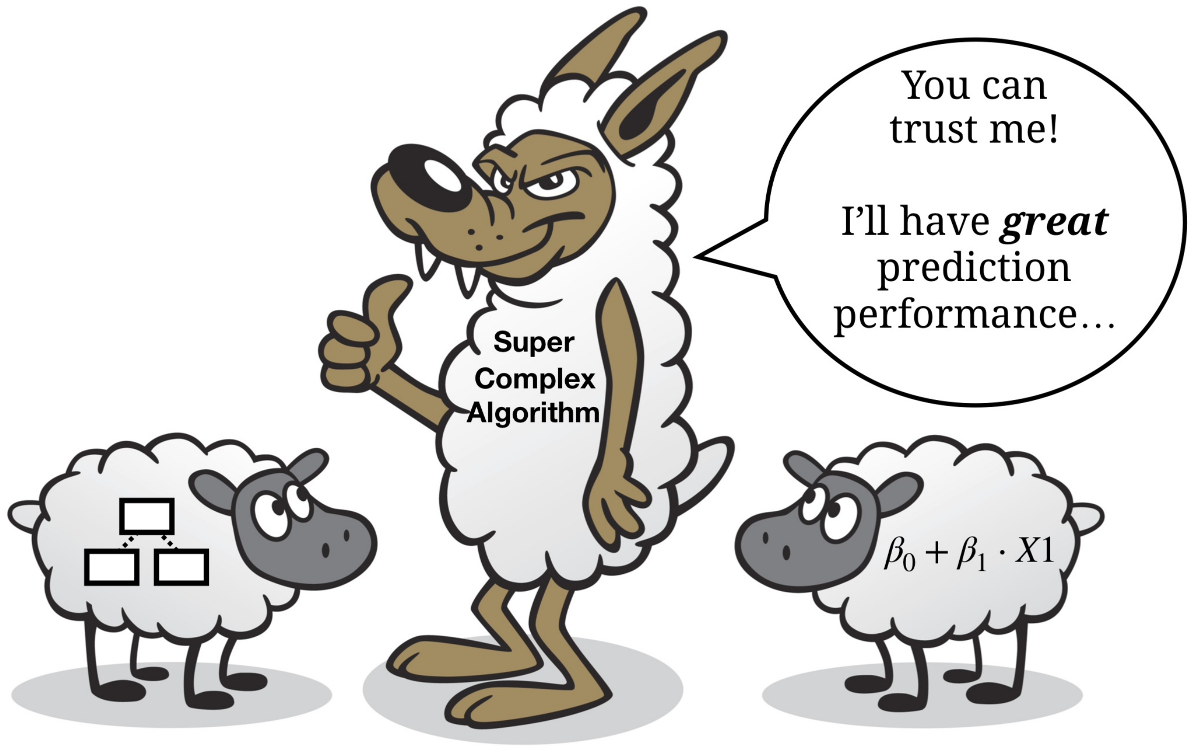
May 2019

Fighting overfitting

When a model **fits the training data too well** on the expense of its performance in prediction, this is called overfitting.

Just because model A is better than model B in training, does not mean it will be better in testing! Extremely flexible models are **'wolves in sheep's clothing'**.

But is there nothing we can do?



adapted from victoriarollison.com

Tuning parameters

All machine learning models are equipped with tuning parameters that **control model complexity.**

These tuning parameters can be identified using a **validation set** created from the traning data.

Logic

1 - Create separate test set.

2 - Fit model using various tuning parameters.

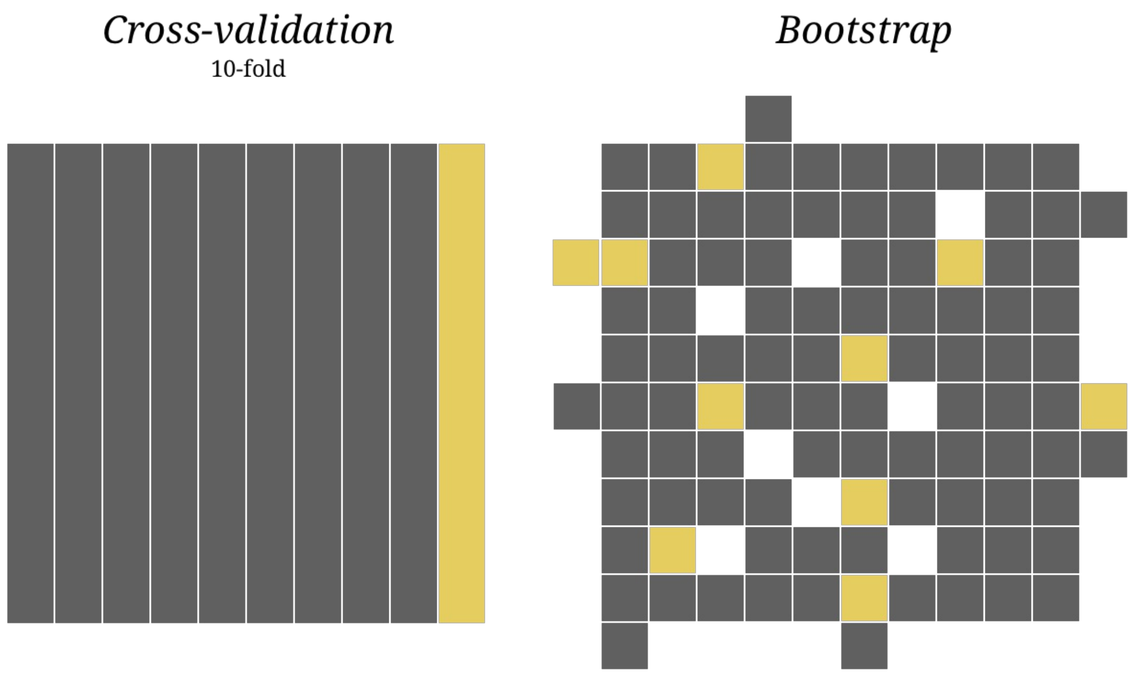
3 - Select tuning leading to best prediction on validation set.

4 - Refit model to entire training set (training + validation).

Resampling methods

Resampling methods automatize and generalize model tuning.

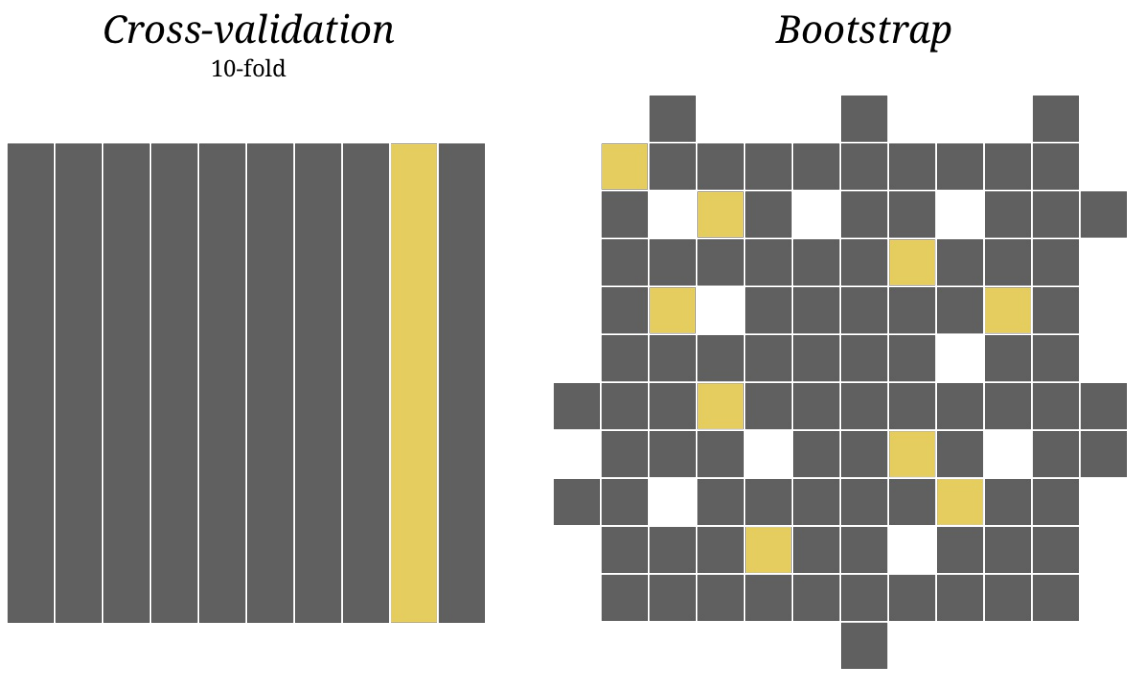
Method	Description
k-fold cross-validation	Splits the data in k-pieces, use **each piece once** as the validation set, while using the other one for training.
Bootstrap	For *B* bootstrap rounds **sample** from the data **with replacement** and split the data in training and validation set.



Resampling methods

Resampling methods automatize and generalize model tuning.

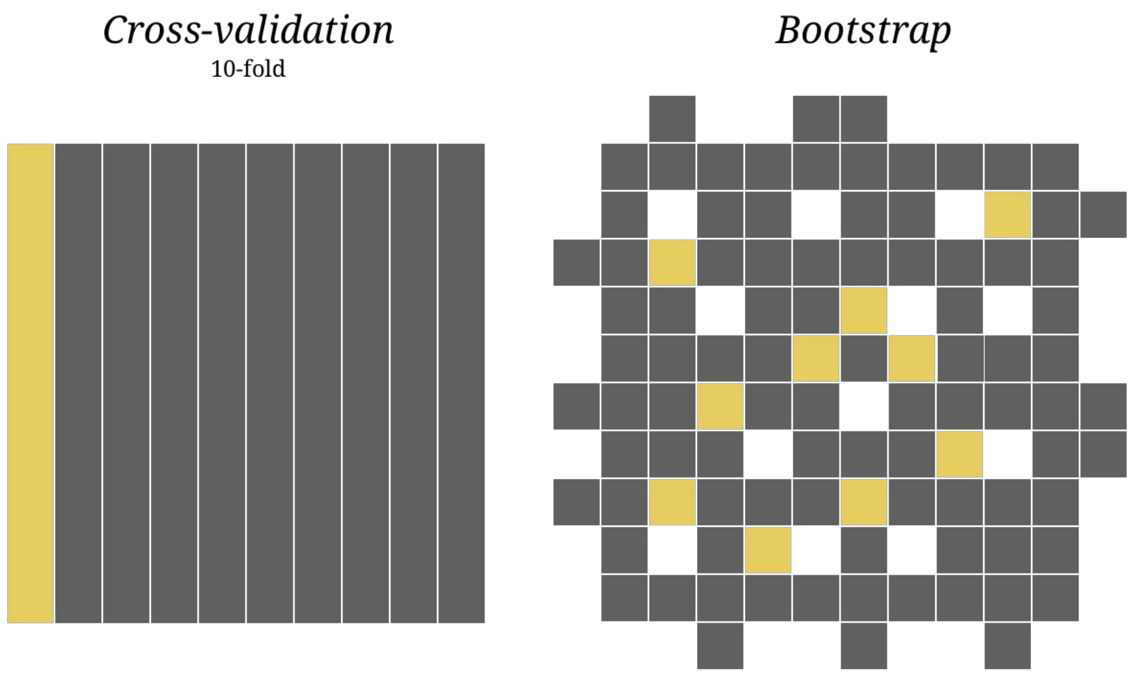
Method	Description
k-fold cross-validation	Splits the data in k-pieces, use **each piece once** as the validation set, while using the other one for training.
Bootstrap	For *B* bootstrap rounds **sample** from the data **with replacement** and split the data in training and validation set.



Resampling methods

Resampling methods automatize and generalize model tuning.

Method	Description
k-fold cross-validation	Splits the data in k-pieces, use **each piece once** as the validation set, while using the other one for training.
Bootstrap	For *B* bootstrap rounds **sample** from the data **with replacement** and split the data in training and validation set.



Regression

Decision Trees

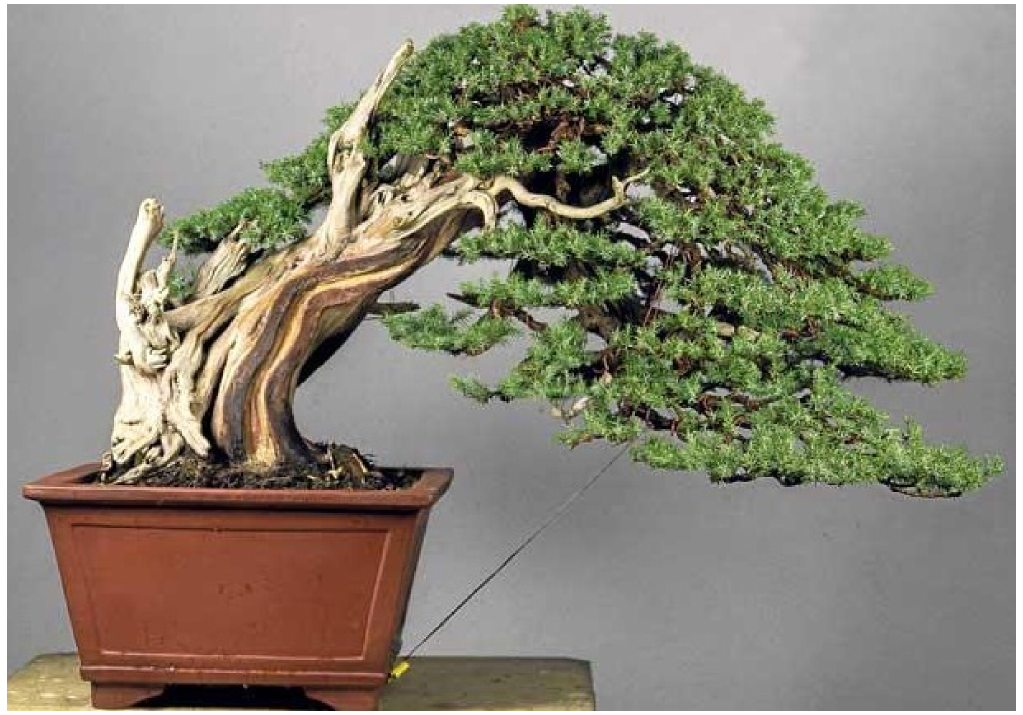
Random Forests

Regularized regression

Penalizes regression loss for having large β values using the **lambda λ tuning parameter** and one of several penalty functions.

$$Regularized\ loss = \sum_{i}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j}^{p}f(\beta_j))$$

Name	**Function**	**Description**
Lasso	$\vert\beta_j\vert$	Penalize by the **absolute** regression weights.
Ridge	β_j^2	Penalize by the **squared** regression weights.
Elastic net	$\vert\beta_j\vert + \beta_j^2$	Penalize by Lasso and Ridge penalties.



from mallorcazeitung.es

Regularized regression

Despite **superficial similarities**, Lasso and Ridge show very different behavior.

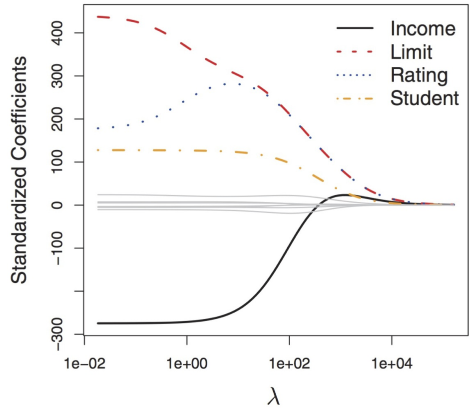
Ridge

By penalizing the most extreme βs most strongly, Ridge leads to (relatively) more **uniform βs**.

Lasso

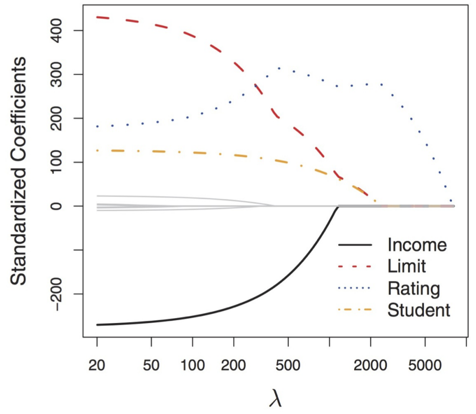
By penalizing all βs equally, irrespective of magnitude, Lasso drives some βs to 0 resulting effectively in **automatic feature selection**.

Ridge



from James et al. (2013) ISLR

Lasso



from James et al. (2013) ISLR

Regularized regression

To fit Lasso or Ridge penalized regression in R, use `method = "glmnet"`.

Specify the **type of penalty** and the **penalty weight** using the `tuneGrid` argument.

<u>`tuneGrid settings`</u>

Parameter	Description
`alpha = 1`	Regression with Lasso penalty.
`alpha = 0`	Regression with Ridge penalty.
`lambda`	Regularization penalty weight.

```
# Train ridge regression
train(form = criterion ~ .,
      data = data_train,
      method = "glmnet",
      trControl = ctrl,
      tuneGrid =
        expand.grid(alpha = 0,    # Ridge
                    lambda = 1)) # Lambda

# Train lasso regression
train(form = criterion ~ .,
      data = data_train,
      method = "glmnet",
      trControl = ctrl,
      tuneGrid =
        expand.grid(alpha = 1,    # Lasso
                    lambda = 1)) # Lambda
```

Regression

Decision Trees

Random Forests

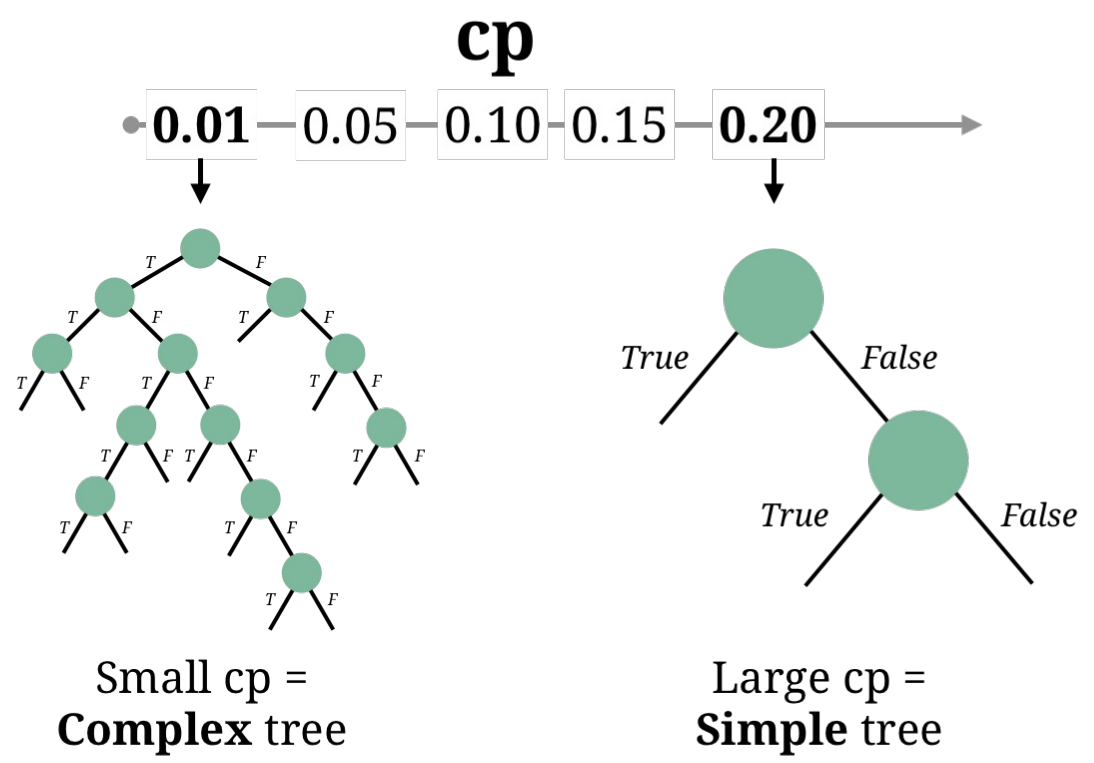
Decision trees

Decision trees have a **complexity parameter** called `cp`.

$$Loss = Impurity + cp * (n\ terminal\ nodes)$$

<u>`tuneGrid` settings</u>

Parameter	Description
Small `cp`, e.g., `cp<.01`	Low penalty leading to **complex trees**.
Large `cp`, e.g., `cp<.20`	Large penalty leading to **simple trees**.



Decision trees

Decision trees have a **complexity parameter** called `cp`.

$$Loss = Impurity + \\ cp * (n\ terminal\ nodes)$$

`tuneGrid` settings

Parameter	Description
Small `cp`, e.g., `cp<.01`	Low penalty leading to **complex trees**.
Large `cp`, e.g., `cp<.20`	Large penalty leading to **simple trees**.

```r
# Decision tree with a defined cp = .01
train(form = income ~ .,
      data = baselers,
      method = "rpart",  # Decision Tree
      trControl = ctrl,
      tuneGrid =
        expand.grid(cp = .01)) # cp

# Decision tree with a defined cp = .2
train(form = income ~ .,
      data = baselers,
      method = "rpart",  # Decision Tree
      trControl = ctrl,
      tuneGrid =
        expand.grid(cp = .2)) # cp
```

Regression

Decision Trees

Random Forests

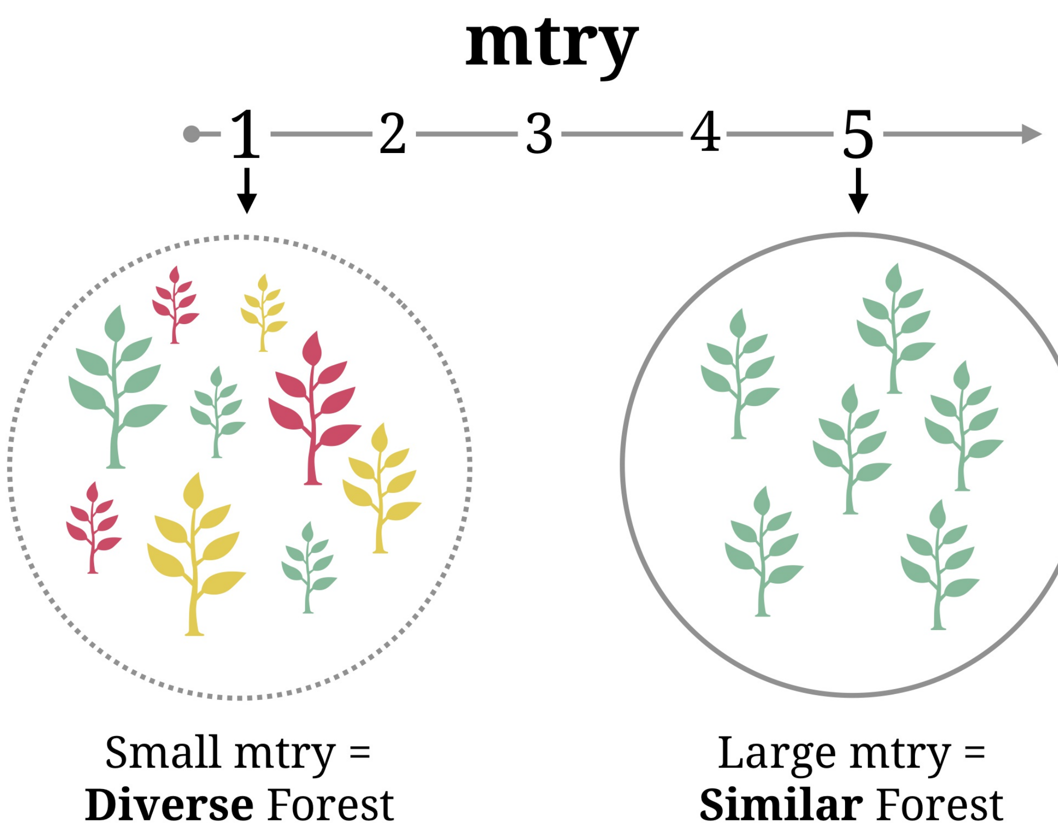
Random Forest

Random Forests have a **diversity parameter** called `mtry`.

Technically, this controls how many features are randomly considered at each split of the trees.

tuneGrid settings

Parameter	Description
Small `mtry`, e.g., `mtry = 1`	**Diverse forest.** In a way, less complex.
Large `mtry`, e.g., `mtry>5`	**Similar forest.** In a way, more complex.



Random Forest

Random Forests have a **diversity parameter** called `mtry`.

Technically, this controls how many features are randomly considered at each split of the trees.

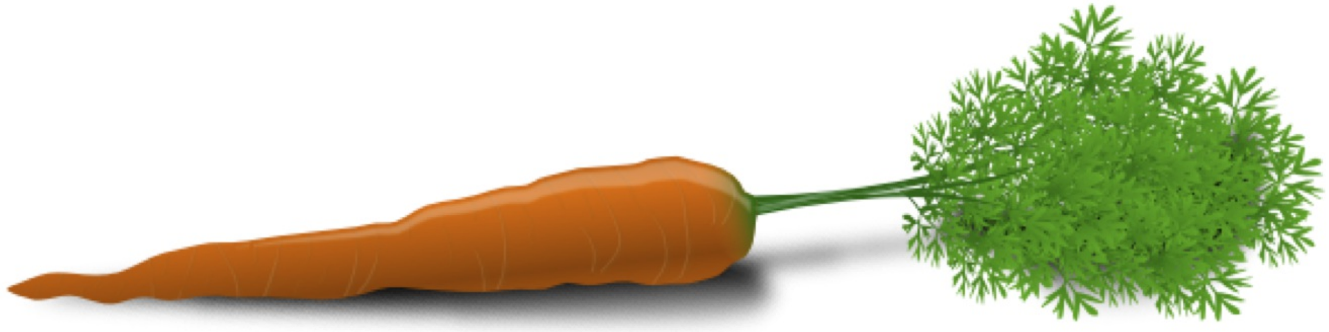
`tuneGrid` settings

Parameter	Description
Small `mtry`, e.g., `mtry = 1`	**Diverse forest.** In a way, less complex.
Large `mtry`, e.g., `mtry>5`	**Similar forest.** In a way, more complex.

```
# Random forest with a defined mtry = 2
train(form = income ~ .,
      data = baselers,
      method = "rf",  # Random forest
      trControl = ctrl,
      tuneGrid =
        expand.grid(mtry = 2)) # mtry

# Random forest with a defined mtry = 5
train(form = income ~ .,
      data = baselers,
      method = "rf",  # Random forest
      trControl = ctrl,
      tuneGrid =
        expand.grid(mtry = 5)) # mtry
```

caret

Parameter tuning with k-fold cross-validation



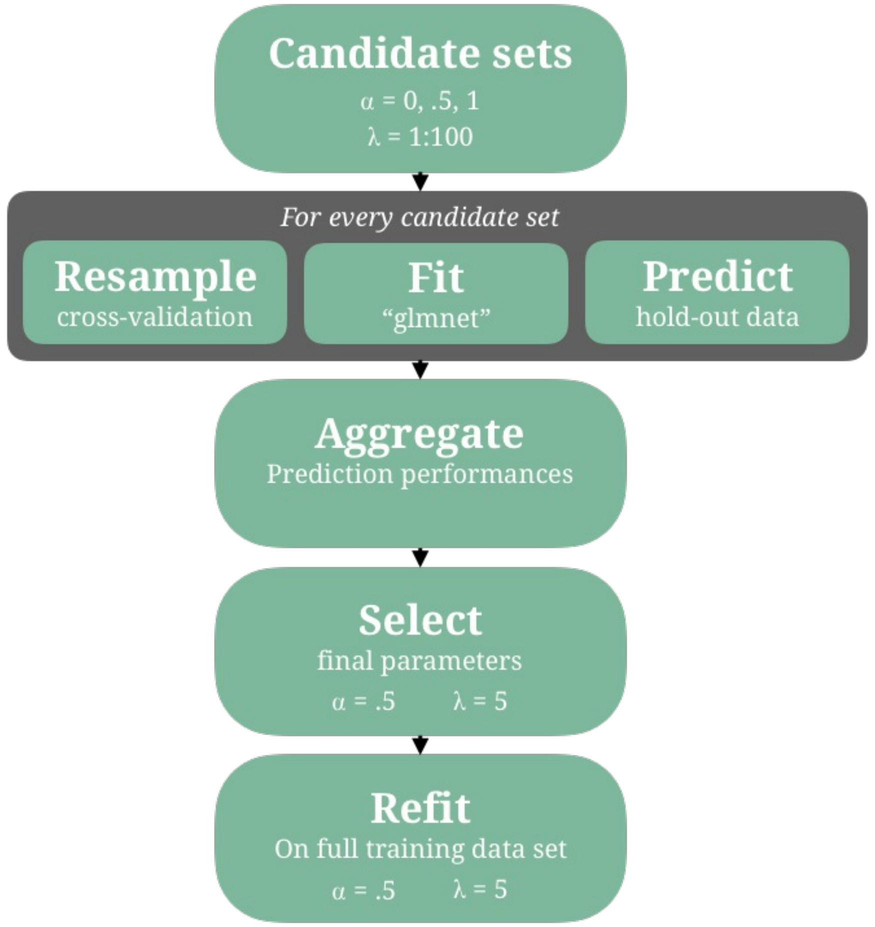
k-fold cross validation for Ridge and Lasso

Goal

Use 10-fold cross-validation to identify **optimal regularization parameters** for a regression model.

Consider

$\alpha \in 0, .5, 1$ and $\lambda \in 1, 2, \ldots, 100$



trainControl()

Specify the use of k-fold cross-validation using the `trainControl()` function.

trainControl() arguments

Argument	Description
`method`	The resampling method, use cv for cross validation.
`number`	The number of folds.

```
# Specify 10 fold cross-validation
ctrl_cv <- trainControl(method = "cv",
                        number = 10)

# Predict income using glmnet
glmnet_mod <- train(form = income ~ .,
                    data = baselers,
                    method = "glmnet",
                    trControl = ctrl_cv)
```

tuneGrid

Specify the tuning parameter values to consider using the `tuneGrid`.

`tuneGrid` expects a **list or data frame** as input.

Parameter combinations can be easily created using `expand.grid`.

```r
# Specify 10 fold cross-validation
ctrl_cv <- trainControl(method = "cv",
                        number = 10)

# Predict income using glmnet
glmnet_mod <- train(form = income ~ .,
                    data = baselers,
                    method = "glmnet",
                    trControl = ctrl_cv,
                    tuneGrid = expand.grid(
                      alpha = c(0, .5, 1),
                      lambda = 1:100))
```

k-Fold Cross validation

```r
# Print summary information
glmnet_mod
```

At the end…

```
RMSE was used to select the optimal
model using the smallest value. The
final values used for the model were
alpha = 1 and lambda = 27.
```

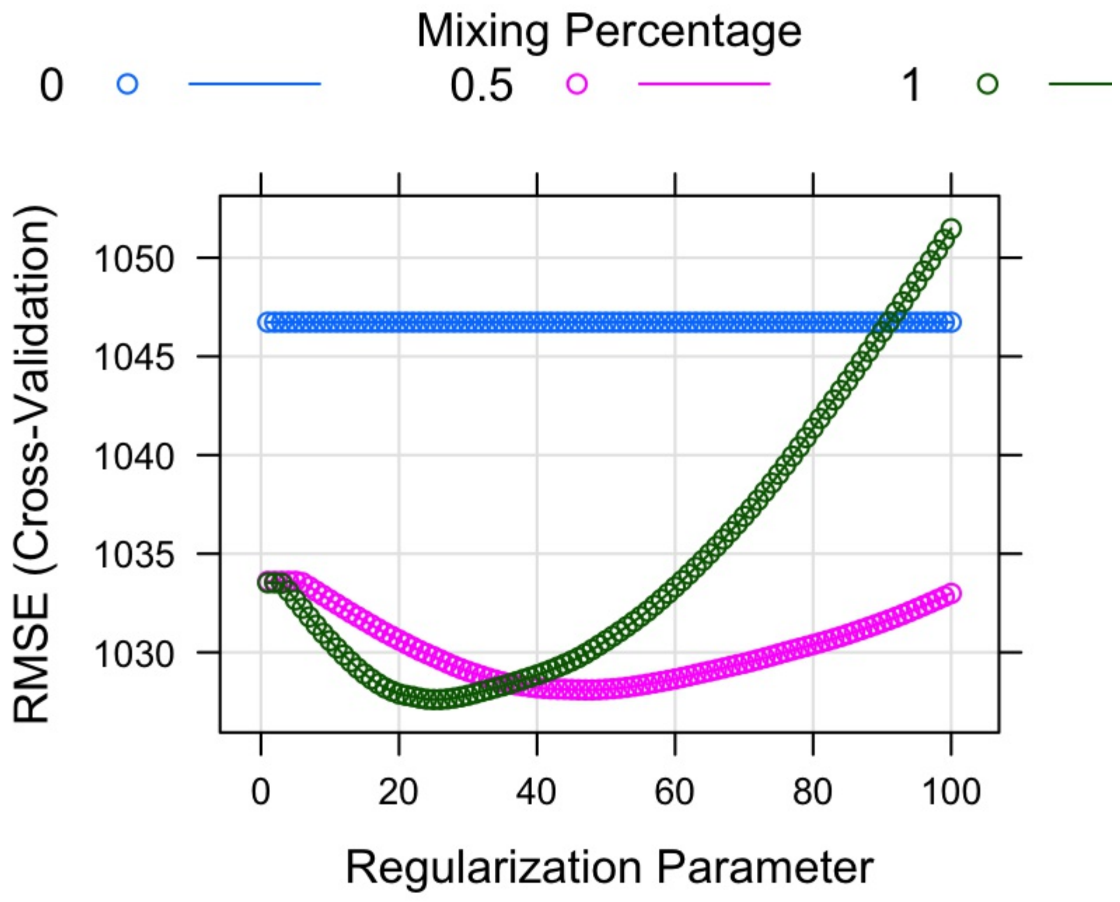
```
glmnet

1000 samples
  19 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 900, 901, 900, 901, 901, 899, ...
Resampling results across tuning parameters:

  alpha  lambda  RMSE  Rsquared  MAE
  0.0      1     1047  0.8614    823.3
  0.0      2     1047  0.8614    823.3
  0.0      3     1047  0.8614    823.3
  0.0      4     1047  0.8614    823.3
  0.0      5     1047  0.8614    823.3
  0.0      6     1047  0.8614    823.3
  0.0      7     1047  0.8614    823.3
  0.0      8     1047  0.8614    823.3
  0.0      9     1047  0.8614    823.3
  0.0     10     1047  0.8614    823.3
  0.0     11     1047  0.8614    823.3
  0.0     12     1047  0.8614    823.3
```

k-Fold Cross validation

```
# Visualise tuning error curve
plot(glmnet_mod)
```

At the end...

```
RMSE was used to select the optimal
model using the smallest value. The
final values used for the model were
alpha = 1 and lambda = 27.
```



Final model

```r
# Model coefficients for best
#   alpha and lambda
coef(glmnet_mod$finalModel,
     glmnet_mod$bestTune$lambda)
```

```
25 x 1 sparse Matrix of class "dgCMatrix"
                                        1
(Intercept)                      462.1958
id                                .
sexmale                           .
age                              116.1387
height                             1.8865
weight                            .
educationobligatory_school        .
educationSEK_II                   .
educationSEK_III                   1.1857
confessionconfessionless           3.9774
confessionevangelical-reformed    .
confessionmuslim                  .
confessionother                   .
children                         -21.0502
happiness                       -128.5448
fitness                           .
food                               2.3193
alcohol                           22.2351
tattoos                          -24.8320
rhine                              0.4256
```

Model comparison

Compare the prediction performance of several models with `resamples()`.

The `summary()` of this object will print 'prediction' error statistics from cross-validation during training. This is your **estimate of future prediction performance**!

```
# Simple competitor model
glm_mod <-  train(form = income ~ .,
                  data = baselers,
                  method = "glm",
                  trControl = ctrl_cv)

# Determine prediction statistics
resamples_mod <- resamples(
  list(glmnet = glmnet_mod,
       glm = glm_mod))

# Print result summary
summary(resamples_mod)
```

Model comparison

Compare the prediction performance of several models with `resamples()`.

The `summary()` of this object will print 'prediction' error statistics from cross-validation during training. This is your **estimate of future prediction performance**!

```
Call:
summary.resamples(object = resamples_mod)

Models: glmnet, glm
Number of resamples: 10

MAE
        Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
glmnet 743.1   761.2  818.3 807.8   836.7 891.7    0
glm    734.8   777.7  801.6 812.8   844.5 892.2    0

RMSE
        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
glmnet 936.5   990.3   1042 1028    1076 1098    0
glm    950.7  1008.9   1016 1034    1063 1128    0

Rsquared
         Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
glmnet 0.8386  0.8440 0.8582 0.8638  0.8865 0.9021    0
glm    0.8268  0.8549 0.8694 0.8624  0.8740 0.8825    0
```

Practical