Resampling and Tuning

Mark Asch - IMU/VLP/CSU

2023

Resampling Methods: Motivation

- Resampling is an indispensable tool of machine learning and modern statistics.

- It provides an evaluation of two important aspects of a general statistical model:

  1. The predictive performance of a given model.
  2. Tuning of the method parameters to find an optimal set.

- It is the best way to avoid data leakage, or data "crimes," where the same data that was used to construct, or train the model, is used to test, or evaluate it.

  $\Rightarrow$ In other words, it is considered bad practice to provide accuracy or predictive performance measures of a statistical method based on the data that was used to estimate the model.

  $\Rightarrow$ We should always attempt an objective evaluation of our models.

⇒ Note that data leakage can also occur during stages of data preparation.

⇒ An example is the use of all the data to perform normalization. One should normalize at the level of the training samples, since otherwise there is information leaking into the model from outside its scope.

⇒ The result will be an overly-optimistic, or overfitted model.

⇒ A model obtained in this way will often be useless in reality.

Resampling Methods: Procedure

- The basic procedure of any resampling method is:

  1. Draw samples, repeatedly, from a training set.
  2. Refit a model on each sample.
  3. Compute an average, or best model.
  4. Apply this model to test sets, which are complementary subsets that have not been used in the training phase.

Resampling Methods: Cross-Validation

- The most commonly-used resampling technique is cross-validation (CV), depicted in the Figure below.

- It is used for:

  $\Rightarrow$ Model evaluation, when the test error associated with a ML method is used to measure its performance.
  $\Rightarrow$ Model selection/tuning, when a suitable level of flexibility of the model is to be chosen by varying its hyperparameters.

- CV has the following forms:

  $\Rightarrow$ k-fold,
  $\Rightarrow$ train/test split ($k = 2$),
  $\Rightarrow$ LOOCV ($k = n$),
  $\Rightarrow$ stratified CV,
  $\Rightarrow$ repeated CV.

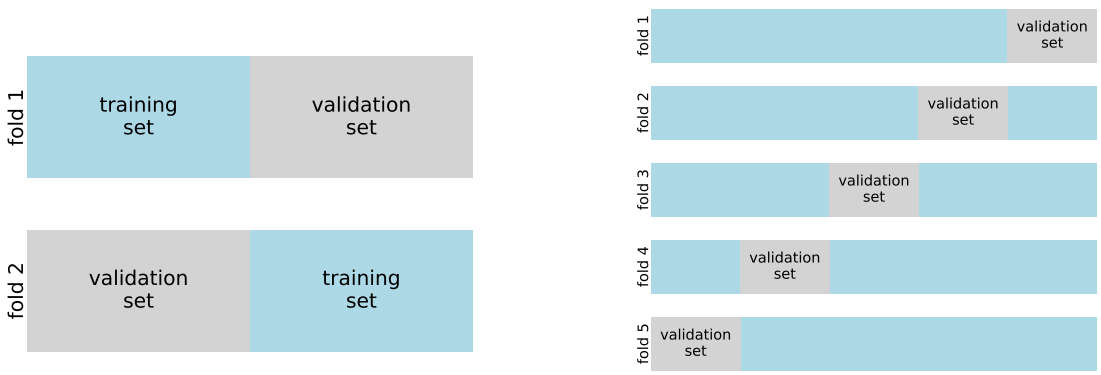


Figure 1: 2-fold (left) and 5-fold CV (right). Training set is blue, validation set is gray.

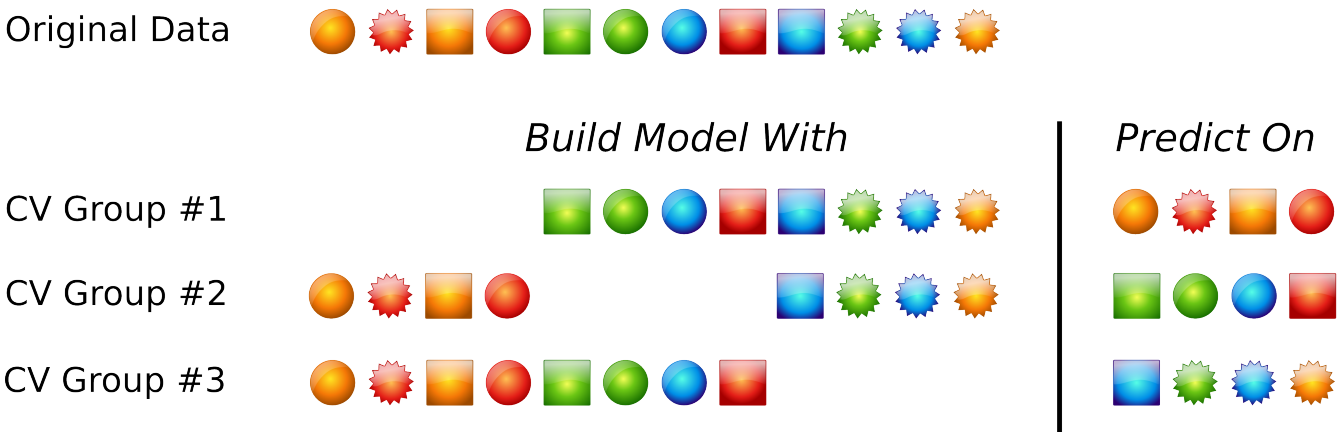
Resampling Methods: Bootstrap

- A second resampling approach is called bootstrap.
- This is used when we want to provide a measure of accuracy, or quantify the uncertainty of any statistical learning method.
- Bootstrap is based on sampling with replacement—see below.

1. CROSS-VALIDATION

Cross-Validation

Definition 1 (Cross-Validation). CV is a class of methods for estimating the test error rate of a statistical model by retaining a subset of the training observations from the fitting process, then applying the computed model to these retained observations.



- The accepted practice for all ML methods is to learn the model on a training set of observations, then test the learned model on an unseen test set of new observations, thus directly estimating the test error rate.

- But often we only have the training set available. To

compensate for this, there are a number of cross-validation techniques that can be employed to estimate the test error rate using the available training data-

- Cross-validation should be used systematically.

1. Cross-Validation: properties

CV has the following properties:

- simple to understand
- relatively straightforward to implement—most ML libraries and routines have built-in CV methods.
- provides an estimation with less bias and potentially less variance, though there is a trade-off—see below.
- provides a less "optimistic" estimation than a simple, single train-test partition of the observation data.

Fact 1. *CV is an essential tool in all statistical learning methods for evaluating the confidence in the model and producing as robust a model as possible that will perform well on future, unseen data.*

1. Cross-Validation: variants

A. Validation Set CV.

This is the simplest form of CV.

- Randomly divide the set of observations into two parts: a training set and a validation set.
- Fit the model using the training set, and use the fitted model to predict the responses for the observations in the validation set.
- The resulting error rate of the validation—typically estimated by the MSE—provides an estimation of the test error rate.

B. Leave-One-Out Cross-Validation (LOOCV)

- A single observation (x_1, y_1) is used for the validation set.

- The remaining observations, $(x_2, y_2), ..., (x_n, y_n)$, constitute the training set.

- Then cycle through the indices, $i = 2, \ldots, n$.

C. k-Fold Cross-Validation

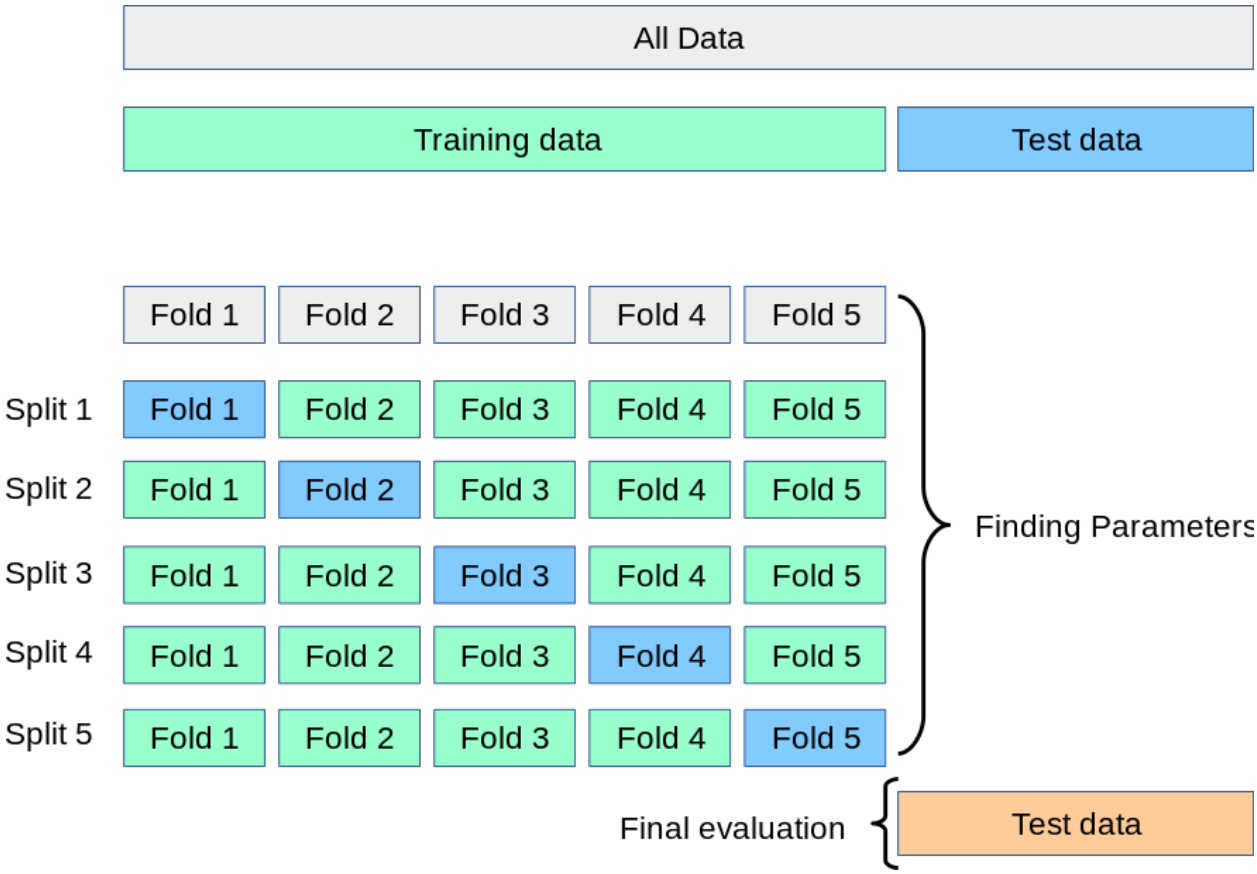
- Randomly divide the set of observations into k groups, called *folds*, of the same size.

- The first group becomes the validation set.

- The method is fit on the $k - 1$ remaining groups.

- The validation set approach is the special case, $k = 2$. LOOCV is the special case, $k = n$.

One usually performs k-fold cross validation with $k = 5$ or $k = 10$, as described in the Algorithm below.


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

CV Algorithm: Train/Test (Validation) Splits

1. Divide the data into two subsets:
   (a) a training set (usually 80%),
   (b) a validation set (usually 20%) that plays the role of a test set.

2. Fit the model to the training subset.

3. Compute the mean-square error of the model applied to the validation subset.

4. Compare with the MSE of the training subset.

5. Modify the sample size, and repeat.

CV Algorithm: k-fold

1. Divide the full data set randomly into k groups, called folds.

2. Fit the data on all the folds, except the first one.

3. Compute the mean squared error on the "left out" fold (the validation set).

4. Repeat the above two steps and compute k estimates of the test error, $\mathrm{MSE}_1, \mathrm{MSE}_2, \ldots, \mathrm{MSE}_k$.

5. Finally, compute the k-fold cross-validation error estimate as the average,

$$\mathrm{CV}_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \mathrm{MSE}_i.$$

Mesures of Performance

- To begin with, we require individual performance scores for each cycle of the CV loop.

  $\Rightarrow$ For regression problems, use the mean squared error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2,$$

  where $\hat{y}_i$ is the predicted value by the model for observation i.

  $\Rightarrow$ For classification problems, use the misclassification rate

$$\text{MCE} = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right),$$

  where $\hat{y}_i$ is the class, or label predicted by the model for observation i.

- Then, the general score of the CV is the average of the

k individual scores,

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{j=1}^{k}\mathrm{MSE}_j,$$

to which we add a measure of the variance, the standard error, which is the standard deviation of the sampling distribution,

$$\mathrm{SE} = \frac{s}{\sqrt{n}},$$

where s is the empirical standard deviation, the square root of the empirical variance,

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2.$$

- Finally, an estimated 95% confidence interval of the empirical mean[1], is defined by

$$\bar{x} \pm 1.96\mathrm{SE}.$$

[1] This confidence interval is exact for a Gaussian distribution, but random i.i.d. errors and their sums are only approximately Gaussian.

Cross-Validation - remarks on k-Fold Cross-Validation

- *Bias-Variance Compromise*: k-fold CV provides more accurate estimates of the test error rate than LOOCV, thanks to the trade-off that causes a smaller bias for LOOCV, but larger variance.

- Any *normalization*, prior to fitting the model, should be performed within the loop and not over all the data, otherwise there is a risk of producing overly optimistic estimations—see the discussion on data leakage below.

- Each observation is affected to a unique group or fold, and remains in its group for the remainder of the procedure.

Cross-Validation - Choice of k

- The value of k must be chosen with care, depending on the context.

- Desired properties are:

  $\Rightarrow$ The value must be *representative*, meaning that k should be chosen so that each train/test group is sufficiently large to contain a representative sample of all the data.

  $\Rightarrow$ $k = 10$ is the value, found by experience, that produces a performance estimation with small bias and modest variance.

  $\Rightarrow$ $k = 5$ is often used to compensate for smaller values of n.

Cross-Validation - Variants of CV

Train/test *split*: one extreme, where $k = 2$.

LOOCV**:** the other extreme, where $k = n$, usable only when n is not too large.

Stratified**:** the split ensures that each fold contains the same proportion of observations of each class.

Repeated**:** repeat the k-fold cross-validation M times, usually 3, 5, or 10, where the data are randomly reshuffled prior to each repetition, thus resulting in different splits each time.

Repeated *Train/test*: divide into train/test, then repeat the division multiple times—this requires a greater number of repetitions than the k-fold method.

✔ Recommended when datasets are small and models are simple.

✔ Produces a more stable estimation of the average performance.

Nested: perform a CV with k-folds within each fold—often used for tuning hyperparameters during model validation.

Cross-Validation -Train-Test Split or k-Fold?

- Train-Test is the simplest version of k-fold CV, with $k = 2$. It is used mainly when n is large and the training cost is high, e.g., in deep neural networks. It is not suited to small values of n for which one should always use classical k-fold. The split proportion is chosen as a function of

  $\Rightarrow$ computational cost of the training,
  $\Rightarrow$ computational cost of model evaluation.
  $\Rightarrow$ representativity of the training set,
  $\Rightarrow$ representativity of the test set.

- The usual values are:

  $\Rightarrow$ train 80%, test 20%,
  $\Rightarrow$ train 67%, test 33%,
  $\Rightarrow$ train 50%, test 50%.

Implementations of CV

There are a few possibilities.

- Code it by hand, either with Python or R—see Example below.
- Use the `caret`[2] package of R—see below for a brief presentation of this extremely useful package.
- Use the `cross_validate`[3] function of `scikit-learn`.

[2] `https://CRAN.R-project.org/package=caret`

[3] `https://scikit-learn.org/stable/modules/cross_validation.html`

2. BOOTSTRAPPING

2. Bootstrap

Definition 2. A bootstrap method is a resampling technique used to estimate population statistics by repeatedly resampling a dataset with *replacement*.

- Bootstrapping has classically been used to estimate summary statistics of a sample, such as means or standard deviations.
- In ML it can be used to estimate the predictive performance on unseen data, as an alternative to CV.
- The chief advantage of bootstrap is that it provides *confidence intervals* for the estimates that are not available from a simple CV.

Fact 2. *Sampling with replacement has the remarkable property that the samples drawn will include approximately* 62% *of the observations only, thus leaving an* out-of-bag *sample, consisting of the approximately* 38% *of unused observations, that can then be used as a test set.*

Bootstrap - confidence intervals

- Bootstrap provides an idea of the confidence interval of an estimation.

- A bootstrap confidence interval, of *level* α, is computed by identifying the percentiles of the bootstrap distribution, leaving on both sides of the distribution $\alpha/2 \times 100\%$.

  $\Rightarrow$ to obtain a good bootstrap confidence interval, the *number of simulations* must be big enough, usually of the order of $N \geq 1000$

- Bootstrap is applicable to a large variety of supervised learning methods, especially when a method does not provide *measures of variability*

Bootstrap - algorithm

1. Choose the number N of bootstrap samples to perform.

2. Choose a sample size n_b from the n available observations, with $n_b \leq n$.

3. For each bootstrap sample:

   (a) Draw a sample with replacement with the chosen size n_b.
   (b) Fit a model on the data sample.
   (c) Estimate the predictive skill of the model on the *out-of-bag* sample.

4. Calculate the mean of the sample of model skill estimates.

5. Calculate a *confidence interval* on the mean.

- The confidence interval can be computed parametrically, or non-parametrically.

⇒ In the former, we suppose an approximate Gaussian sampling distribution and use the standard $\mathcal{N}(0, 1)$ confidence limits.

⇒ The non-parametric approach simply requires the ordering of the values, followed by the identification of the chosen quantiles according to the confidence level chosen.

Bootstrap - remarks

Some important remarks are:

- The bootstrap is applicable to most statistical learning methods, especially when there are no measures of uncertainty available.

- Due to the sampling with replacement, a bootstrapped dataset may contain multiple instances of the same original observations, and may completely omit other original observations.

- To obtain a reliable confidence interval, a large number of simulations must be performed, usually $N > 1000$. This will not be feasible for computationally expensive models.

- Though one usually takes the bootstrap sample size n_b equal to the totality of the available samples, n, this can be reduced to 50% or 80% of the original dataset in the case where n is very large.

Implementations of Bootstrap

The function calls are:

R

```
library(boot)
bootobject <- boot(data= , statistic= , R=N, ...)
```

scikit

```
from sklearn.utils import resample
boot = resample(data, replace=True,
        n_samples=N, random_state=1)
```

Cross Validation vs. Bootstrapping

- Both cross validation and bootstrapping are resampling methods.

  $\Rightarrow$ Cross-Validation provides estimates of the test error.
  $\Rightarrow$ Bootstrap provides, in addition, the standard error of the estimates.

- As the name cross-validation suggests, its primary purpose is measuring (generalization) performance of a model.

- By contrast, bootstrapping is primarily used to establish empirical distribution functions for a widespread range of statistics, ranging from the variation of the mean to the variation of models in bagged ensemble models.

- The bootstrap equivalent to cross-validation estimates of the predictive performance error is called an out-of-bootstrap, or out-of-bag (OOB) estimate.

Remark 1. In practice there is often not much of a difference between iterated k-fold cross-validation and out-of-bootstrap. With a similar total number of evaluated sampling models, the model prediction error measurement is similar, although OOB typically has more bias and less variance than the corresponding CV estimates.

Resampling: Take-Home Lessons

No resampling method is uniformly better than another and the choice of a particular method should consider several aspects.

- If the sample size is small, repeated 10-fold cross-validation is recommended for several reasons: the bias and variance properties are good and, given the sample size, the computational costs are not large.

- If the goal is to choose between models, as opposed to getting the best indicator of performance, it is advisable to use a bootstrap method since these have very low variance.

- For large sample sizes, the differences between resampling methods become less pronounced, and computational efficiency becomes the critical factor. Here, simple 10-fold cross-validation should provide acceptable variance and low bias, and is relatively cheap to compute.

The R command `boot`

- the pacakge `boot` provides numerous possibilities for obtaining bootstrap estimations

- command and arguments:

  $\Rightarrow$ `bootobject <- boot(data = , statistic = , R = , ...)`

  $\rightarrow$ `data` is a vector, matrix or dataframe

  $\rightarrow$ `statistic` is a function that gives the k statistics to be bootdtrapped; the function must include a parameter/argument `indices` so that `boot` can choose the cases for each replica

  $\Rightarrow$ `R` the number of repeats of the - normaly around 1000 (at least)

- `boot()` calls the function «statistic» R times, each time using sampling with replacement, and the statistics are computed and stored in the structure `bootobject` that contains

⇒ `t0` are the observed values of the k statistics applied to the original data

⇒ `t` is a matrix $R \times k$, where each row is a bootstrap replicaof the k statistics

- outputs:

  ⇒ `print(bootobjet)`

  ⇒ `plot(bootobjet)`

- confidence intervals: if the outputs are acceptable, then the confidence intervals can be calculated by

  ⇒ `boot.ci(bootobjet, conf = , type = )`

  → `conf` is the level (by default 0.95)

  → `type` is the distribution: 'norm', 'basic', 'stud', 'perc', 'bca' and 'all' (by default 'all')

3. MODEL TUNING

3. Model Tuning

- Most ML methods require the choice of (hyper)parameters that cannot be estimated directly from the data.

- For example,

  $\Rightarrow$ in the k-nn classification algorithm, we need to choose the number of neighbors;
  $\Rightarrow$ for SVM with a radial kernel, we must choose the exponent;
  $\Rightarrow$ for neural networks, the complete architecture—layers and nodes—needs to be chosen, as well as the SGD optimization parameters.

- These *tuning parameters* control the model complexity, and a bad choice will lead to overfitting—see below.

3. Model Tuning - general approach

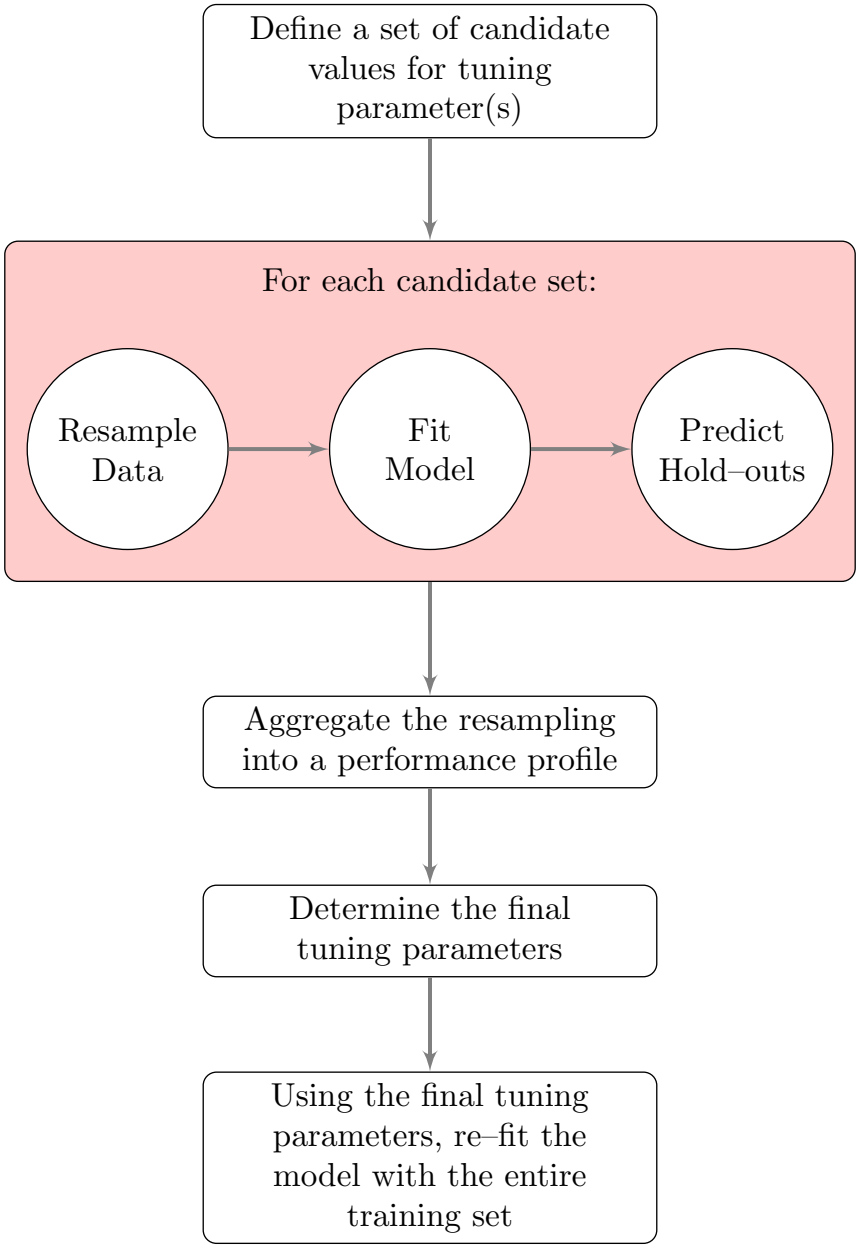
- To deal with this thorny problem, a general approach that can be applied to almost any model is:

  ⇒ Define a set of candidate values for the parameter(s).
  ⇒ Calculate robust estimations of the performance, usually using CV.
  ⇒ Choose the optimal parameter value(s).

- Then, to obtain robust performance estimates:

  ⇒ Divide the date into training and testing sets according to one of the resampling methods and variants seen above.
  ⇒ Choose a candidate set, or range of values for the parameters—often, tuning methods will provide default ranges, but these might need to be modified in a given context.
  ⇒ Test the performance over the test set.
  ⇒ Aggregate the results to draw a performance profile.
  ⇒ Select the best model.

⇒ Apply the final model to all the data, using the tuning parameters that have just been selected.



3. Model Tuning - final parameter choice

- Once the model performance has been evaluated over the parameter sets, there are different approaches for the final choice of the tuning parameters.

  $\Rightarrow$ The simplest is to choose the parameters that gave the best performance, usually based on the average and its standard error.

  $\Rightarrow$ But, to avoid overly complex models, the one standard error method can be used to choose a simpler model:

  $\rightarrow$ Identify the optimal model.

  $\rightarrow$ Compute its standard error.

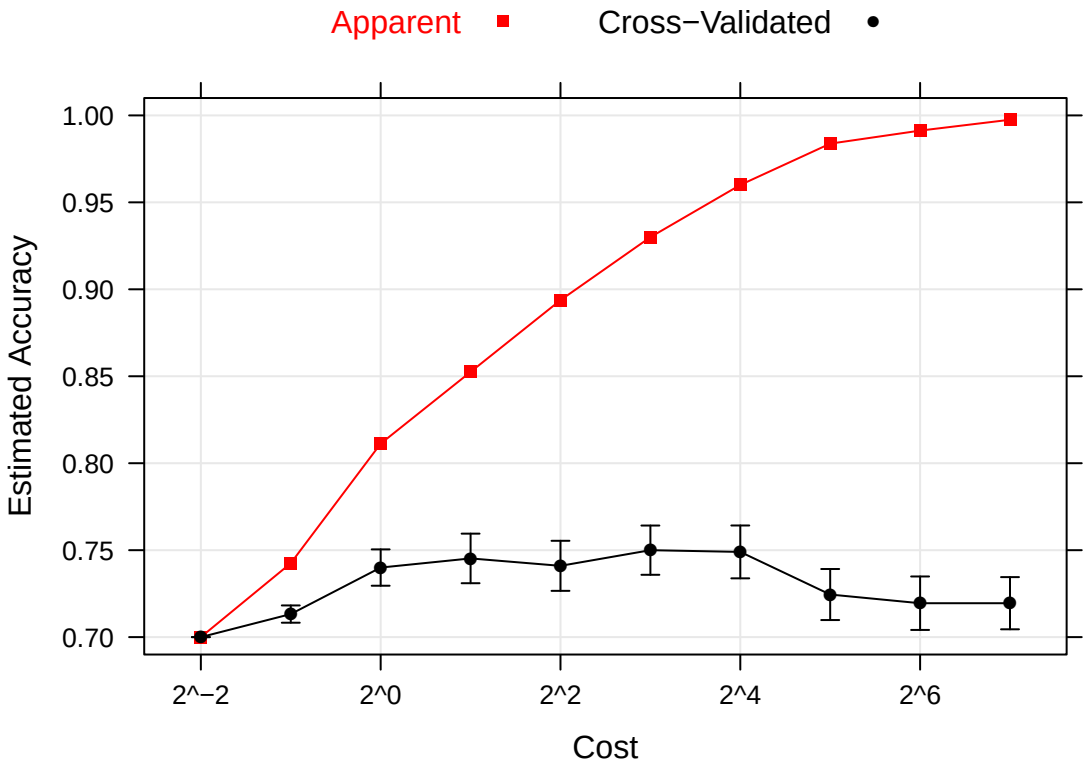
  $\rightarrow$ Find the *simplest* model within an interval of one standard error.

- A final approach is to specify a small, acceptable tolerance of the best value and then choose the simplest model within this tolerance.

- Example of final choices:

  $\Rightarrow$ simplest approach is to take the best performance estimations

  $\rightarrow$ here we have 5 repetitions of 10-fold CV, for a total of 50 estimations of the precision, then we compute the average and its standard error $= s/\sqrt{n}$, where s is the empirical standard deviation)



- But, to avoid overly complex models:

$\Rightarrow$ the «one-standard error» method enables the choice of a simpler model

- $\rightarrow$ identify the optimal model (cost = 8)
- $\rightarrow$ calculate its standard error
- $\rightarrow$ find the simplest model in an interval of one standard error from the optimal value—here we will choose the model with cost = 2

The `caret` package

- `caret` = **C**lassifcation **A**nd **RE**gression **T**raining
- provides a uniform/homogeneous interface to a vast diversity of models (~ 240...) for their construction and their prediction
- there are also commands that facilitate parallel computing...
- collection of fonctions to help in the creation of predictive models with tools for:
  - ⇒ division/partition/ splitting of data
    - → `sample` (R)
    - → `createDataPartition()`
  - ⇒ transformations and normalization of data
    - → `preProcess()`
    - → `dummyVars()`
  - ⇒ tuning of models by resampling

$\rightarrow$ `train()`

$\Rightarrow$ feature selection

$\Rightarrow$ estimation of variable importance

- Remark : in scikit-learn we find similar functions...

Commands of `caret` package

- we procede in teh following order:
  - ⇒ `trainControl()` : define a CV method
  - ⇒ `expand.grid()` : define a grid for tuning
  - ⇒ `train()` : perform the training
  - ⇒ `print/plot()` : print/display the results

- the command `train()` :

```
> model <- train(formula,
                 data = " ",
                 trControl = " ",
                 tuneGrid = " ",
                 method = " ")
```

- each method has its own, specific tuning parameters - consult the documentation:

- `https://topepo.github.io/caret/available-models.html`

The package `tidymodels`

- can replace `caret`
- based on the packages `tidyverse`
- uses a synatax of «pipelines» just as in `scikit-learn`
- can use the library `keras` for intensive ML on big data

Examples

1. CV with repetitions for NN : CV.html
2. CV with caret for NB : CV_caret.html
3. Tuning with caret for SVM : SVM_caret_Pima.html
4. Bootstrap for car data (SLR) : boot_cars.html

References

1. M. Kuhn, K. Johnson. *Applied Predictive Modeling*. Springer, 2013.

2. M. DeGroot, M. Schervish, *Probability and Statistics*, Addison Wesley, 2002.

3. Spiegel, Murray and Larry Stephens, *Schaum's Outline of Statistics,* 6th edition, McGraw Hill. 2017.

4. G. James, D. Witten, T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning with Applications in R.* Springer. 2013.

5. T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning*. Springer. 2009.

6. Rachel Schutt and Cathy O'Neil. *Doing Data Science.* O'Reilly. 2014.