Goals of an Evaluation Method

- The golden rule:
  
  *The data used for evaluating (or comparing) any models cannot be seen during model development.*

- The goal of any evaluation procedure:
  - Obtain a reliable estimate of some evaluation measure.  
    *High probability of achieving the same score on other samples of the same population.*

- Evaluation Measures
  - Predictive accuracy.
  - Model size.
  - Computational complexity.
The usual techniques for model evaluation revolve around resampling.
- Simulating the reality.
  - Obtain an evaluation estimate for unseen data.

Examples of Resampling-based Methods
- Holdout.
- Cross-validation.
- Bootstrap.

Time Series Data Are Special!
Any form of resampling changes the natural order of the data!

General Guidelines
- Do not “forget” the time tags of the observations.
- Do not evaluate a model on past data.

A possible method
- Divide the existing data in two time windows
  - Past data (observations till a time \( t \)).
  - “Future” data (observations after \( t \)).
- Use one of these three learn-test alternatives
  - Fixed learning window.
  - Growing window.
  - Sliding window.
Learn-Test Strategies

**Fixed Window**

A single model is obtained with the available “training” data, and applied to all test period.

**Growing Window**

Every $w_v$ test cases a new model is obtained using all data available till then.

**Sliding Window**

Every $w_v$ test cases a new model is obtained using the previous $w_s$ observations of the time series.

---

Dealing with model selection

- Most modelling techniques involve some form of parameters that usually need to be tuned.
- The following describes an evaluation methodology considering this issue:

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$\cdots$</th>
<th>$y_s$</th>
<th>$\cdots$</th>
<th>$y_t$</th>
<th>$\cdots$</th>
<th>$y_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1</td>
<td>Data used for obtaining the model alternatives</td>
<td>Model tuning and selection period</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage 2</td>
<td>Data used for obtaining the selected model alternative / variant</td>
<td>Final Evaluation Period</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some Metrics for Evaluating Predictive Performance

**Absolute Measures**

- **Mean Squared Error (MSE)**
  \[ MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{x}_i - x_i)^2 \]

- **Mean Absolute Deviation (MAD)**
  \[ MAD = \frac{1}{n} \sum_{i=1}^{n} |\hat{x}_i - x_i| \]

**Relative Measures**

- **Theil Coefficient**
  \[ U = \frac{\sqrt{\sum_{i=1}^{n} (\hat{x}_i - x_i)^2}}{\sqrt{\sum_{i=1}^{n} (x_i - x_{i-1})^2}} \]

- **Mean Absolute Percentage Error (MAPE)**
  \[ MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{\hat{x}_i - x_i}{x_i} \right| \]

---

The Metrics in R

```r
set.seed(1234)
library(xts)
someSeries <- xts(rnorm(1000), seq.Date(from = Sys.Date(), length.out = 1000, by = "1 day"))
somePreds <- xts(rnorm(1000), seq.Date(from = Sys.Date(), length.out = 1000, by = "1 day"))
(mse <- mean((someSeries - somePreds)^2))
## [1] 1.846431

(mad <- mean(abs(someSeries - somePreds)))
## [1] 1.070021

(U <- sqrt(sum(((someSeries - somePreds)^2)[-1])) / sqrt(sum(((someSeries - lag(someSeries, 1))^2)[-1])))
## [1] 0.9771318

(mape <- mean(abs((someSeries - somePreds)/someSeries)))
## [1] 4.551463
```
The Goal of an Experimental Comparison

- Given a set of observations of a time series $X$.
- Given a set of alternative modelling approaches $M$.
- Obtain estimates of the predictive performance of each $m_i$ for this time series.

More specifically,
- given a forecasting period size, $w_{test}$,
- and a predictive performance statistic, $Err$,
we want to obtain a reliable estimate of the value of $Err$
for each $m_i$.

Using Monte Carlo Simulations for Obtaining Reliable Estimates of $Err$

- A possible approach would be to use our proposed method of Model Selection.
- This would give us one estimate of $Err$.
- More reliability is achievable if more repetitions of the process are carried out.

Monte Carlo Estimates for Time Series Forecasting

Given: a time series, a training window size, $w_{train}$, a testing window size, $w_{test}$, and a number of repetitions, $r$,
- randomly generate $r$ points in the interval $]w_{train}..(n - w_{test})[$,
- for each point proceed according to our Model Selection strategy.
Experimental Comparisons  The Goals

Using Monte Carlo Simulations for Obtaining Reliable Estimates of $\text{Err} - 2$

Available Time Series Data

The Goals

Using Monte Carlo Simulations for Obtaining Reliable Estimates of $\text{Err} - 2$

The Infra-Structure of package performanceEstimation

- The package `performanceEstimation` provides a set of functions that can be used to carry out comparative experiments of different models on different predictive tasks.
- This infra-structure can be applied to any model/task/evaluation metric.
- Installation:
  - Official release (from CRAN repositories):
    ```r
    install.packages("performanceEstimation")
    ```
  - Development release (from Github):
    ```r
    library(devtools)  # You need to install this package before!
    install_github("ltorgo/performanceEstimation", ref="develop")
    ```
The main function of the package is `performanceEstimation()`. It has 3 arguments:

1. The predictive tasks to use in the comparison
2. The models to be compared
3. The estimation task to be carried out

The function implements a wide range of experimental methodologies including all we have discussed.

A Simple Example

Suppose we want to estimate the mean squared error of regression trees in a certain regression task using cross validation.

```r
library(performanceEstimation)
library(DMwR)
data(Boston, package='MASS')
res <- performanceEstimation(
    PredTask(medv ~ ., Boston),
    Workflow("standardWF", learner="rpartXse"),
    EstimationTask(metrics="mse", method=CV(nReps=1, nFolds=10)))
```
### Summary of a Cross Validation Performance Estimation Experiment

Task for estimating \( \text{mse} \) using 1 x 10 - Fold Cross Validation

Run with seed = 1234

* Predictive Tasks :: Boston.medv
* Workflows :: rpartXse

-> Task: Boston.medv
* Workflow: rpartXse

- \( \text{mse} \)
- \( \text{avg} \) = 19.610531
- \( \text{std} \) = 9.375305
- \( \text{med} \) = 16.867969
- \( \text{iqr} \) = 11.523275
- \( \text{min} \) = 9.266761
- \( \text{max} \) = 34.752888
- invalid = 0.000000

---

### Plot of Cross Validation Performance Estimation Results

Distribution of Statistics Scores

Cross Validation Performance Estimation Results

Alternative Workflows

© L.Torgo (FCUP - LIAAD / UP)  
TS Evaluation  
Jan, 2017  
15 / 41
Predictive Tasks

- Objects of class **PredTask** describing a predictive task
  - Classification
  - Regression
  - Time series forecasting
- Created with the constructor with the same name

```r
data(iris)
PredTask(Species ~ ., iris)
```

```r
## Prediction Task Object:
## Task Name :: iris.Species
## Task Type :: classification
## Target Feature :: Species
## Formula :: Species ~ .
## Task Data Source :: iris

PredTask(Species ~ ., iris,"IrisDS",copy=TRUE)
```

## Prediction Task Object:
## Task Name :: IrisDS
## Task Type :: classification
## Target Feature :: Species
## Formula :: Species ~ .
## Task Data Source :: internal 150x5 data frame.

Workflows

- Objects of class **Workflow** describing an approach to a predictive task
  - Standard Workflows
    - Function `standardWF` for classification and regression
    - Function `timeseriesWF` for time series forecasting
  - User-defined Workflows
Standard Workflows for Classification and Regression Tasks

```
library(e1071)
Workflow("standardWF", learner="svm", learner.pars=list(cost=10, gamma=0.1))

## Workflow Object:
## Workflow ID :: svm
## Workflow Function :: standardWF
## Parameter values:
## learner -> svm
## learner.pars -> cost=10 gamma=0.1
```

“standardWF” can be omitted ...

```
Workflow(learner="svm", learner.pars=list(cost=5))

## Workflow Object:
## Workflow ID :: svm
## Workflow Function :: standardWF
## Parameter values:
## learner -> svm
## learner.pars -> cost=5
```

Main parameters of the constructor:

- **Learning stage**
  - `learner` - which function is used to obtain the model for the training data
  - `learner.pars` - list with the parameter settings to pass to the learner

- **Prediction stage**
  - `predictor` - function used to obtain the predictions (defaults to `predict()`)  
  - `predictor.pars` - list with the parameter settings to pass to the predictor
Standard Workflows for Classification and Regression Tasks (cont.)

- Main parameters of the constructor (cont.):
  - Data pre-processing
    - `pre` - vector with function names to be applied to the training and test sets before learning
    - `pre.pars` - list with the parameter settings to pass to the functions
  - Predictions post-processing
    - `post` - vector with function names to be applied to the predictions
    - `post.pars` - list with the parameter settings to pass to the functions

```r
data(algae, package="DMwR")
res <- performanceEstimation(
  PredTask(a1 ~ ., algae[,1:12], "A1"),
  Workflow(learner="lm", pre="centralImp", post="onlyPos"),
  EstimationTask("mse", method=CV())  # defaults to 1x10-fold CV
)
```

## ###### PERFORMANCE ESTIMATION USING CROSS VALIDATION ######

## ** PREDICTIVE TASK :: A1 ##

## ++ MODEL/WORKFLOW :: lm ##

## Task for estimating mse using
## 1 x 10 - Fold Cross Validation
## Run with seed = 1234
## Iteration :**********
Evaluating Variants of Workflows

Function `workflowVariants()`

```r
library(e1071)
data(Boston, package="MASS")
res2 <- performanceEstimation(
    PredTask(medv ~ ., Boston),
    workflowVariants(learner="svm",
        learner.pars=list(cost=1:5, gamma=c(0.1, 0.01))),
    EstimationTask(metrics="mse", method=CV()))
```

---

```
summary(res2)
```

```r
## == Summary of a Cross Validation Performance Estimation Experiment ==
## Task for estimating mse using
## 1 x 10 - Fold Cross Validation
## Run with seed = 1234
## * Predictive Tasks :: Boston.medv
## * Workflows :: svm.v1, svm.v2, svm.v3, svm.v4, svm.v5, svm.v6, svm.v7, svm.v8, svm.v9, svm.v10
## --> Task: Boston.medv
##   *Workflow: svm.v1
##     mse
##     avg 14.80685
##     std 10.15295
##     med 12.27015
##     iqr 11.87737
##     min 5.35198
##     max 38.39681
##     invalid 0.00000
##   *Workflow: svm.v2
##     mse
##     avg 11.995178
##     std 7.908371
##     med 8.359433
##     iqr 11.626306
##     min 4.842848
##     max 28.480351
##     invalid 0.00000
##   *Workflow: svm.v3
##     mse
##     avg 11.045068
##     std 7.014775
##     med 7.185975
##     iqr 10.693513
##     min 4.421629
##     max 24.199194
##     invalid 0.00000
##   *Workflow: svm.v4
##     mse
##     avg 10.773223
##     std 6.684297
##     med 7.147570
##     iqr 10.088544
##     min 4.364334
##     max 23.082813
##     invalid 0.00000
##   *Workflow: svm.v5
##     mse
##     avg 10.650186
##     std 6.489002
##     med 7.406310
##     iqr 9.664462
##     min 4.304427
##     max 22.870107
##     invalid 0.00000
##   *Workflow: svm.v6
##     mse
##     avg 18.832056
##     std 11.033421
##     med 16.086489
##     iqr 15.678784
##     min 6.716207
##     max 40.813201
##     invalid 0.00000
##   *Workflow: svm.v7
##     mse
##     avg 16.530852
##     std 10.333326
##     med 13.753694
##     iqr 14.738903
##     min 6.144193
##     max 37.115054
##     invalid 0.00000
##   *Workflow: svm.v8
##     mse
##     avg 15.483001
##     std 10.027918
##     med 12.141047
##     iqr 13.340164
##     min 6.020978
##     max 35.789052
##     invalid 0.00000
##   *Workflow: svm.v9
##     mse
##     avg 14.988977
##     std 9.923290
##     med 11.289272
##     iqr 12.788812
##     min 6.101292
##     max 34.914317
##     invalid 0.00000
##   *Workflow: svm.v10
##     mse
##     avg 14.598540
##     std 9.777895
##     med 10.957707
##     iqr 12.188397
##     min 6.069622
##     max 33.883953
##     invalid 0.00000
```
Exploring the Results

```r
getWorkflow("svm.v1", res2)

## Workflow Object:
## Workflow ID :: svm.v1
## Workflow Function :: standardWF
## Parameter values:
## learner.pars -> cost=1 gamma=0.1
## learner -> svm

topPerformers(res2)

## $Boston.medv
## Workflow Estimate
## mse svm.v5 10.65
```
Estimation Tasks

- Objects of class **EstimationTask** describing the estimation task
  - Main parameters of the constructor
    - `metrics` - vector with names of performance metrics
    - `method` - object of class **EstimationMethod** describing the method used to obtain the estimates

```
EstimationTask(metrics=c("F","rec","prec"), method=Bootstrap(nReps=100))
```

## Task for estimating F, rec, prec using 100 repetitions of e0 Bootstrap experiment
## Run with seed = 1234

Performance Metrics

- Many classification and regression metrics are available
  - Check the help page of functions `classificationMetrics` and `regressionMetrics`
- User can provide a function that implements any other metric she/he wishes to use
  - Parameters `evaluator` and `evaluator.pars` of the `EstimationTask` constructor
Comparing Different Algorithms on the Same Task

```r
library(randomForest)
library(e1071)
res3 <- performanceEstimation(
    PredTask(medv ~ ., Boston),
    workflowVariants("standardWF",
        learner=c("rpartXse","svm","randomForest")),
    EstimationTask(metrics="mse", method=CV(nReps=2, nFolds=5)))
```

Some auxiliary functions

```r
rankWorkflows(res3, 3)
```

```r
# $Boston.medv
# $Boston.medv$mse
# Workflow Estimate
# 1 randomForest 10.87221
# 2 svm 14.89183
# 3 rpartXse 19.73468
```
The Results

```
plot(res3)
```

An example using Holdout and a classification task

```r
data(Glass, package='mlbench')
res4 <- performanceEstimation(
  PredTask(Type ~ ., Glass),
  workflowVariants(learner="svm", # You may omit "standardWF"!
    learner.pars=list(cost=c(1,10),
                       gamma=c(0.1,0.01))),
  EstimationTask(metrics="err", method=Holdout(nReps=5, hldSz=0.3)))
```
The Results

```
plot(res4)
```

An example involving more than one task

```
data(Glass, package='mlbench')
data(iris)
res5 <- performanceEstimation(
  c(PredTask(Type ~ ., Glass), PredTask(Species ~ ., iris)),
  c(workflowVariants(learner="svm",
      learner.pars=list(cost=c(1,10),
        gamma=c(0.1,0.01))),
    workflowVariants(learner="rpartXse",
      learner.pars=list(se=c(0,0.5,1)),
        predictor.pars=list(type="class"))),
  EstimationTask(metrics="err", method=CV(nReps=3))
)```

© L.Torgo (FCUP - LIAAD / UP)  TS Evaluation  Jan, 2017  34 / 41
The Results

```
plot(res5)
```

```
topPerformers(res5)
```

```
## $Glass.Type
## Workflow Estimate
## err  svm.v1  0.294
##
## $iris.Species
## Workflow Estimate
## err  svm.v2  0.04
```

```
topPerformer(res5,"err","Glass.Type")
```

```
## Workflow Object:
##  Workflow ID      :: svm.v1
##  Workflow Function :: standardWF
## Parameter values:
##  learner.pars -> cost=1 gamma=0.1
##  learner  -> svm
```
An example involving time series

First getting the data and building an illustrative data set

```r
library(quantmod)
library(lubridate)
getSymbols("GOOGL", from=Sys.Date() - years(5))

## [1] "GOOGL"

gg <- Delt(Cl(GOOGL))
library(DMwR2)
library(TTR)

dat <- createEmbedDS(gg, emb=7)
dat <- data.frame(cbind(lag(gg,-1),
                     dat,
                     MA10=SMA(gg,10),
                     RSI=RSI(gg),
                     BB=BBands(gg)$pctB))

colnames(dat)[1] <- "FutureT"

dat <- na.omit(dat)
```

Now comparing models

```r
library(e1071)
library(randomForest)

tsExp <- performanceEstimation(
    PredTask(FutureT ~ ., dat,'GG'),
    c(Workflow('timeseriesWF',wfID="slideSVM",
               type="slide",relearn.step=90,
               learner='svm',learner.pars=list(cost=10,gamma=0.01)),
    Workflow('timeseriesWF',wfID="slideRF",
              type="slide",relearn.step=90,
              learner='randomForest',learner.pars=list(ntrees=500))
),
    EstimationTask(metrics="theil",
                   method=MonteCarlo(nReps=10,szTrain=0.5,szTest=0.25))
)
Checking the results

```r
summary(tsExp)
```

```r
#-- Summary of a Monte Carlo Performance Estimation Experiment --#
# Task for estimating theil using
# 10 repetitions Monte Carlo Simulation using:
# seed = 1234
# train size = 0.5 x NROW(DataSet)
# test size = 0.25 x NROW(DataSet)
#
#  Task: GG
#  Workflow: slideSVM, slideRF
#
#  Task: GG
#  Workflow: slideSVM
#   theil
#    avg 0.96309293
#    std 0.04438200
#    med 0.94650894
#    iqr 0.06204175
#    min 0.92142661
#    max 1.04377023
#    invalid 0.00000000
#
#  Task: GG
#  Workflow: slideRF
#   theil
#    avg 1.02660932
#    std 0.08274638
#    med 1.00318257
#    iqr 0.02447508
#    min 0.94601480
#    max 1.21653003
#    invalid 0.00000000
```

Checking the results - 2

```r
plot(tsExp)
```
Hands on Performance Estimation
the Algae data set

Load in the data set `algae` from package `DMwR` and answer the following questions:

1. Estimate the MSE of a regression tree for forecasting alga `a1` using 10-fold Cross validation.
2. Repeat the previous exercise this time trying some variants of random forests. Check what are the characteristics of the best performing variant.
3. Compare the results in terms of mean absolute error of the default variants of a regression tree, a linear regression model and a random forest, in the task of predicting alga `a3`. Use 2 repetitions of a 5-fold Cross Validation experiment.