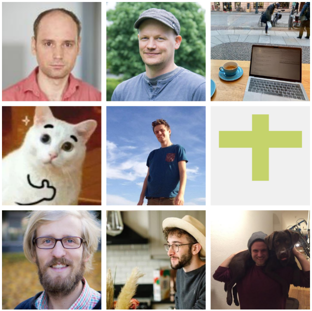
Hyperparameter Tuning with Bayesian Optimization: mlr3mbo

`https://mlr-org.com/`

`https://github.com/mlr-org`

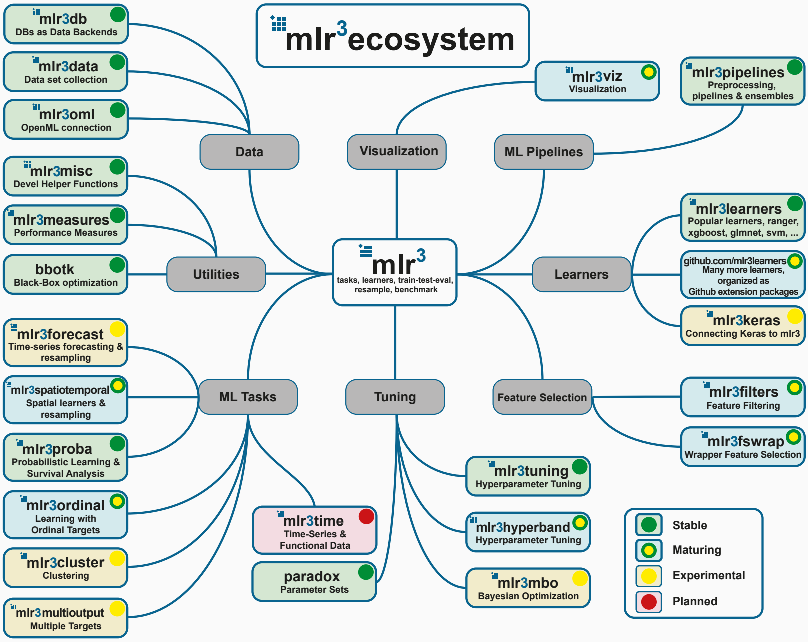


Jakob Richter, Marc Becker, Michel Lang, Martin Binder, Bernd Bischl and more!
LMU Munich / TU Dortmund, Germany
September 24, 2020

Structure

1. `mlr3` basics
2. Bayesian Optimization
3. `mlr3tuning`
4. `mlr3mbo` + `mlr3tuning` (exercise!)

mlr3verse



mlr3 ecosystem
mlr3db
DBs as Data Backends
mlr3data
Data set collection
mlr3oml
OpenML connection
Data
Visualization
mlr3viz
Visualization
ML Pipelines
mlr3pipelines
Preprocessing, pipelines & ensembles
mlr3misc
Devel Helper Functions
mlr3measures
Performance Measures
bbotk
Black-Box optimization
Utilities
mlr3
tasks, learners, train-test-eval, resample, benchmark
Learners
mlr3learners
Popular learners, ranger, xgboost, glmnet, svm, ...
github.com/mlr3learners
Many more learners, organized as Github extension packages
mlr3keras
Connecting Keras to mlr3
mlr3forecast
Time-series forecasting & resampling
mlr3spatiotemporal
Spatial learners & resampling
mlr3proba
Probabilistic Learning & Survival Analysis
mlr3ordinal
Learning with Ordinal Targets
mlr3cluster
Clustering
mlr3multioutput
Multiple Targets
ML Tasks
Tuning
Feature Selection
mlr3filters
Feature Filtering
mlr3fswrap
Wrapper Feature Selection
mlr3time
Time-Series & Functional Data
paradox
Parameter Sets
mlr3tuning
Hyperparameter Tuning
mlr3hyperband
Hyperparameter Tuning
mlr3mbo
Bayesian Optimization
Stable
Maturing
Experimental
Planned

Intro

So you want to do ML in R

- R gives you access to many machine learning methods

So you want to do ML in R

- R gives you access to many machine learning methods
- ...but without a unified interface

So you want to do ML in R

- R gives you access to many machine learning methods
- ...but without a unified interface
- things like performance evaluation are cumbersome

So you want to do ML in R

- R gives you access to many machine learning methods
- ...but without a unified interface
- things like performance evaluation are cumbersome

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

So you want to do ML in R

- R gives you access to many machine learning methods
- …but without a unified interface
- things like performance evaluation are cumbersome

Example:

```
# Specify what we want to model in a formula: target ~ features
svm_model = e1071::svm(Species ~ ., data = iris)
```

vs.

```
# Pass the features as a matrix and the target as a vector
xgb_model = xgboost::xgboost(data = as.matrix(iris[1:4]),
  label = iris$Species, nrounds = 10)
```

So you want to do ML in R

```r
library("mlr3")
```

Ingredients:

- Data / Task
- Learning Algorithms
- Performance Evaluation
- Performance Comparison

R6

R6 – All You Need to Know

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

R6 – All You Need to Know

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow
## [1] 150
```

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow
## [1] 150
```

- Objects have *methods* that are called like functions:

```
task$filter(rows = 1:10)
```

`mlr3` uses the *R6* class system. Some things may seem unusual if you see them for the first time.

- *Objects* are created using `<Class>$new()`.

```
task = TaskClassif$new("iris", iris, "Species")
```

- Objects have *fields* that contain information about the object.

```
task$nrow
## [1] 150
```

- Objects have *methods* that are called like functions:

```
task$filter(rows = 1:10)
```

- Methods may change (“mutate”) the object (reference semantics)!

```
task$nrow
## [1] 10
```

R6 and Active Bindings

Some fields of R6-objects may be *"Active Bindings"*. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11
## Error: Field/Binding is read-only
```

R6 and Active Bindings

Some fields of R6-objects may be *"Active Bindings"*. Internally they are realized as functions that are called whenever the value is set or retrieved.

- Active bindings for read-only fields

```
task$nrow = 11
## Error: Field/Binding is read-only
```

- Active bindings for argument checking

```
task$properties = NULL
## Error in assert_set(rhs, .var.name = "properties"): Assertion on
'properties' failed: Must be of type 'character', not 'NULL'.
task$properties = c("property1", "property2")  # works
```

mlr3 Philosophy

- Overcome limitations of S3 with the help of R6
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics

mlr3 Philosophy

- Overcome limitations of S3 with the help of R6
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics
- Embrace data.table, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure

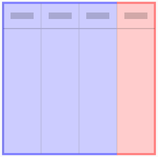
mlr3 Philosophy

- Overcome limitations of S3 with the help of R6
  - Truly object-oriented: data and methods live in the same object
  - Make use of inheritance
  - Reference semantics
- Embrace data.table, both for arguments and internally
  - Fast operations for tabular data
  - List columns to arrange complex objects in tabular structure
- Be light on dependencies:
  - `R6`, `data.table`, `lgr`, `uuid`, `mlbench`, `digest`
  - Plus some of our own packages (`backports`, `checkmate`, …)

Data

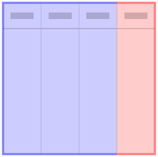
Data

- Tabular data



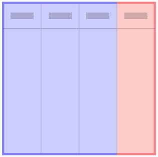
Data

- Tabular data
- Features



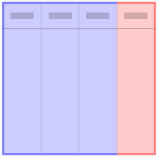
Data

- Tabular data
- Features
- Target / outcome to predict



Data

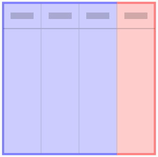
- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression



Data

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

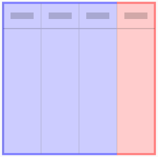
  ⇒ target determines the machine learning "Task"



Data

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning "Task"

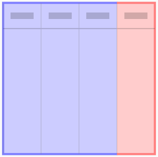


```r
print(iris)  # included in R

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## ...
```

Data

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning “Task”

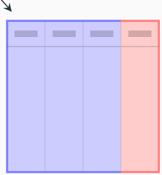


```
print(iris)  # included in R

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## ...
```

Data

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning "Task"



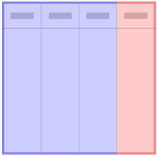
```r
print(iris)  # included in R

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## ...
```

```r
task = TaskClassif$new("iris", iris, "Species")
```

Data

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning “Task”



```
print(iris)  # included in R

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## ...
```

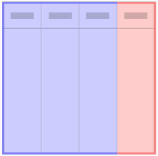
Task ID

```
task = TaskClassif$new("iris", iris, "Species")
```

Data

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning "Task"



```r
print(iris)  # included in R
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## ...
```

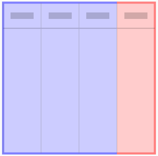
Task ID data

```r
task = TaskClassif$new("iris", iris, "Species")
```

Data

- Tabular data
- Features
- Target / outcome to predict
  - discrete for classification
  - continuous for regression

  ⇒ target determines the machine learning "Task"



```
print(iris)  # included in R

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## ...
```

Task ID   data   target name

```
task = TaskClassif$new("iris", iris, "Species")
```

Data

```r
task = TaskClassif$new("iris", iris, "Species")
```

```r
print(task)

# <TaskClassif:iris> (150 x 5)
# * Target: Species
# * Properties: multiclass
# * Features (4):
#   - dbl (4): Petal.Length, Petal.Width, Sepal.Length, Sepal.Width
```

```r
task$ncol
task$nrow
task$feature_names
task$target_names
```

```r
task$head(n = )
task$truth(row_ids = )
task$data(rows = ,
          cols = )
```

```r
task$select(cols = )
task$filter(rows = )
task$cbind(data = )
task$rbind(data = )
```

Dictionaries

Dictionaries

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

Dictionaries

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

⇒ `mlr3` offers *Short Form Constructors* that are less verbose

Dictionaries

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

⇒ `mlr3` offers *Short Form Constructors* that are less verbose

- They access `Dictionary` of objects:

Dictionaries

- Ordinary constructors: `TaskClassif$new()` / `LearnerClassifRpart$new()`

⇒ `mlr3` offers *Short Form Constructors* that are less verbose

- They access `Dictionary` of objects:

Object	Dictionary	Short Form
`Task`	`mlr_tasks`	`tsk()`
`Learner`	`mlr_learners`	`lrn()`
`Measure`	`mlr_measures`	`msr()`
`Resampling`	`mlr_resamplings`	`rsmp()`

Dictionaries can get populated by add-on packages (e.g. `mlr3learners`)

Dictionaries

```
# list items
tsk()

## <DictionaryTask> with 10 stored values
## Keys: boston_housing, breast_cancer, german_credit, iris, mtcars, pima,
##   sonar, spam, wine, zoo

# retrieve object
tsk("iris")

## <TaskClassif:iris> (150 x 5)
## * Target: Species
## * Properties: multiclass
## * Features (4):
##   - dbl (4): Petal.Length, Petal.Width, Sepal.Length, Sepal.Width
```

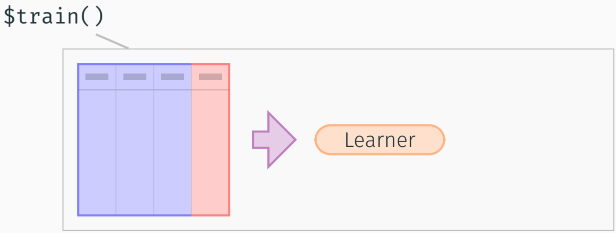
Short Forms and Dictionaries

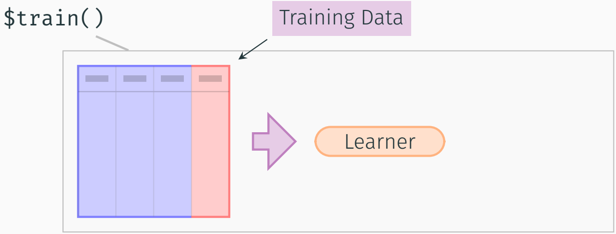
`as.data.table(<DICTIONARY>)` creates a `data.table` with metadata about objects in dictionaries:

```
mlr_learners_table = as.data.table(mlr_learners)

mlr_learners_table[1:10, c("key", "packages", "predict_types")]

#                      key packages predict_types
#                   <char>   <list>        <list>
#  1:        classif.debug            response,prob
#  2: classif.featureless            response,prob
#  3:        classif.rpart    rpart response,prob
#  4:     regr.featureless    stats   response,se
#  5:           regr.rpart    rpart      response
#  6:                 <NA>
#  7:                 <NA>
#  8:                 <NA>
#  9:                 <NA>
# 10:                 <NA>
```

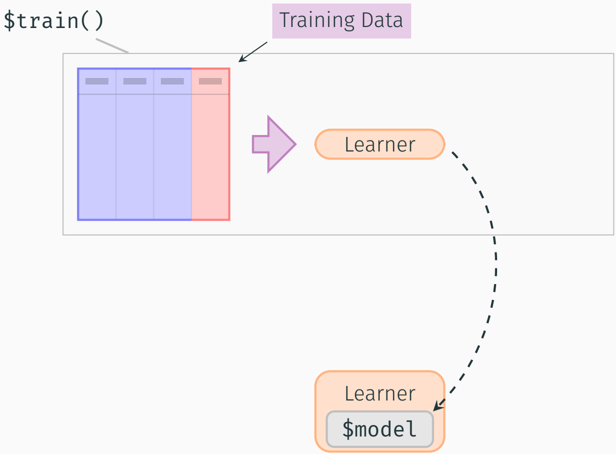
Learning Algorithms

Learning Algorithms

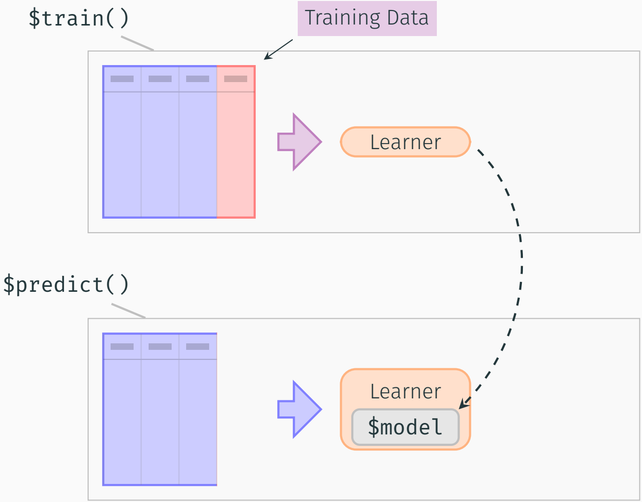



$train()
Training Data
Learner

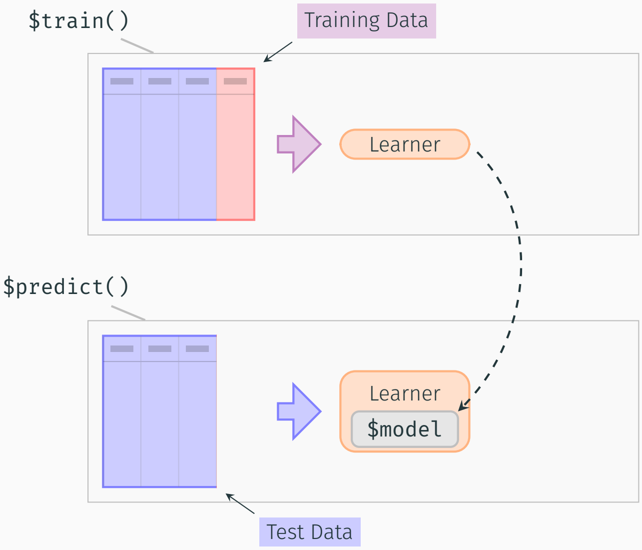
Learning Algorithms

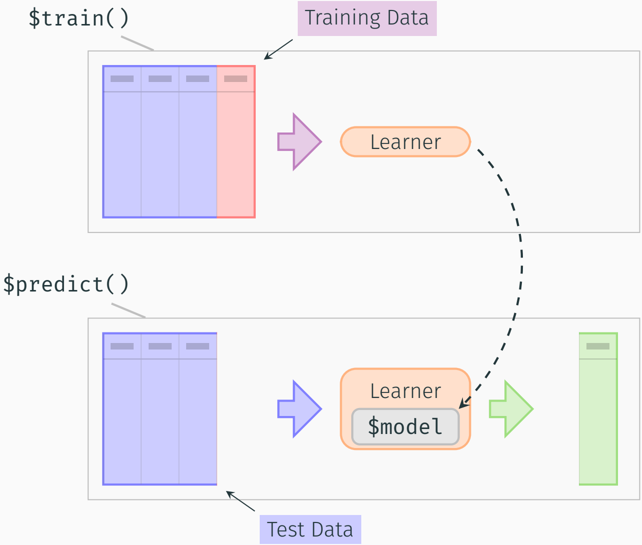


Learning Algorithms

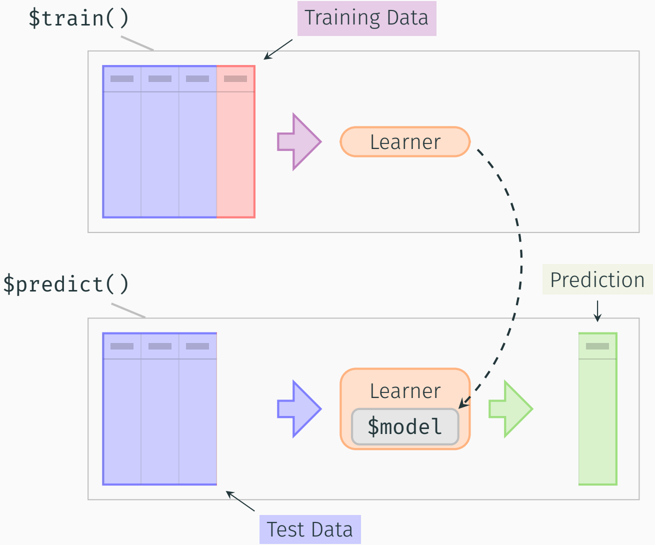


Learning Algorithms




$train()
Training Data
Learner
$predict()
Learner
$model
Test Data

Learning Algorithms



Learning Algorithms

- Get a `Learner` provided by `mlr`

```r
learner = lrn("classif.rpart")
```

Learning Algorithms

- Get a `Learner` provided by `mlr`

```
learner = lrn("classif.rpart")
```

- Train the `Learner`

```
learner$train(task)
```

Learning Algorithms

- Get a `Learner` provided by `mlr`

```
learner = lrn("classif.rpart")
```

- Train the `Learner`

```
learner$train(task)
```

- The `$model` is the `rpart` model: a decision tree

```
print(learner$model)

## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 150 100 setosa (0.333 0.333 0.333)
##   2) Petal.Length< 2.5 50   0 setosa (1.000 0.000 0.000) *
##   3) Petal.Length>=2.5 100  50 versicolor (0.000 0.500 0.500)
##     6) Petal.Width< 1.8 54   5 versicolor (0.000 0.907 0.093) *
##     7) Petal.Width>=1.8 46   1 virginica (0.000 0.022 0.978) *
```

Hyperparameters

- Learners have *hyperparameters*

```
as.data.table(learner$param_set)[, 1:6]

##                  id    class lower upper      levels nlevels
##              <char>   <char> <num> <num>      <list>   <num>
##  1:        minsplit ParamInt     1   Inf                 Inf
##  2:       minbucket ParamInt     1   Inf                 Inf
##  3:              cp ParamDbl     0     1                 Inf
##  4:      maxcompete ParamInt     0   Inf                 Inf
##  5:    maxsurrogate ParamInt     0   Inf                 Inf
##  6:        maxdepth ParamInt     1    30                  30
##  7:    usesurrogate ParamInt     0     2                   3
##  8: surrogatestyle ParamInt     0     1                   2
##  9:            xval ParamInt     0   Inf                 Inf
## 10:      keep_model ParamLgl    NA    NA  TRUE,FALSE       2
```

Hyperparameters

- Learners have *hyperparameters*

```
as.data.table(learner$param_set)[, 1:6]

##                  id    class lower upper      levels nlevels
##              <char>   <char> <num> <num>      <list>   <num>
##  1:        minsplit ParamInt     1   Inf                 Inf
##  2:       minbucket ParamInt     1   Inf                 Inf
##  3:              cp ParamDbl     0     1                 Inf
##  4:      maxcompete ParamInt     0   Inf                 Inf
##  5:    maxsurrogate ParamInt     0   Inf                 Inf
##  6:        maxdepth ParamInt     1    30                  30
##  7:    usesurrogate ParamInt     0     2                   3
##  8: surrogatestyle ParamInt     0     1                   2
##  9:            xval ParamInt     0   Inf                 Inf
## 10:      keep_model ParamLgl    NA    NA  TRUE,FALSE       2
```

- Changing them changes the `Learner` behavior

```
learner$param_set$values = list(maxdepth = 1, xval = 0)

learner$train(task)
```

Hyperparameters

- This gives a smaller decision tree

```
print(learner$model)

## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 150 100 setosa (0.33 0.33 0.33)
##   2) Petal.Length< 2.5 50   0 setosa (1.00 0.00 0.00) *
##   3) Petal.Length>=2.5 100  50 versicolor (0.00 0.50 0.50) *
```

Prediction

- Let's make a prediction for some new data, e.g.:

```
new_data
```

```
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1            4           3            2           1
# 2            2           2            3           2
```

Prediction

- Let's make a prediction for some new data, e.g.:

```
new_data
```

```
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1            4           3            2           1
# 2            2           2            3           2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

Prediction

- Let's make a prediction for some new data, e.g.:

```
new_data
```

```
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1            4           3            2           1
# 2            2           2            3           2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction
```

```
## <PredictionClassif> for 2 observations:
##  row_id truth   response
##       1  <NA>     setosa
##       2  <NA> versicolor
```

Prediction

- Let's make a prediction for some new data, e.g.:

```
new_data
```

```
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1            4           3            2           1
# 2            2           2            3           2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction
```

```
## <PredictionClassif> for 2 observations:
##  row_id truth   response
##       1  <NA>     setosa
##       2  <NA> versicolor
```

Prediction

- Let's make a prediction for some new data, e.g.:

```
new_data
```

```
#   Sepal.Length Sepal.Width Petal.Length Petal.Width
# 1            4           3            2           1
# 2            2           2            3           2
```

- To do so, we call the `$predict_newdata()` method using the new data:

```
prediction = learner$predict_newdata(new_data)
```

- We get a `Prediction` object:

```
prediction
```

```
## <PredictionClassif> for 2 observations:
##  row_id truth   response
##       1  <NA>     setosa
##       2  <NA> versicolor
```

Prediction

- We can make the `Learner` predict *probabilities* when we set `predict_type`:

```
learner$predict_type = "prob"
learner$predict_newdata(new_data)
```

```
# <PredictionClassif> for 2 observations:
#  row_id truth  response prob.setosa prob.versicolor
#       1  <NA>    setosa           1             0.0
#       2  <NA> virginica           0             0.5
#  prob.virginica
#             0.0
#             0.5
```

Prediction

What exactly is a `Prediction` object?

- Contains predictions and offers useful access fields / methods

Prediction

What exactly is a `Prediction` object?

- Contains predictions and offers useful access fields / methods

⇒ Use `as.data.table()` to extract data

```
as.data.table(prediction)

##    row_id  truth   response
##     <int> <fctr>     <fctr>
## 1:      1   <NA>     setosa
## 2:      2   <NA> versicolor
```

Prediction

What exactly is a `Prediction` object?

- Contains predictions and offers useful access fields / methods

$\Rightarrow$ Use `as.data.table()` to extract data

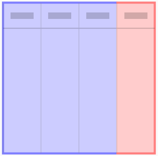
```
as.data.table(prediction)

##    row_id  truth   response
##     <int> <fctr>     <fctr>
## 1:      1   <NA>     setosa
## 2:      2   <NA> versicolor
```

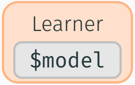
$\Rightarrow$ Active bindings and functions that give further information: `$response`, `$truth`, ...

```
prediction$response

## [1] setosa     versicolor
## Levels: setosa versicolor virginica
```

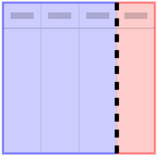
Performance

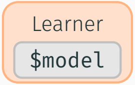
Performance Evaluation



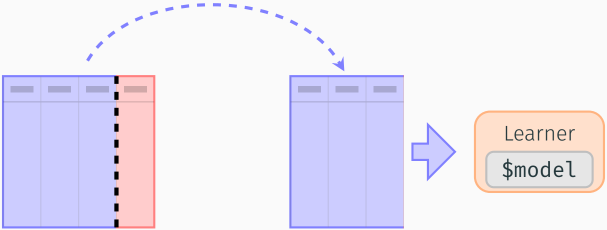


Performance Evaluation

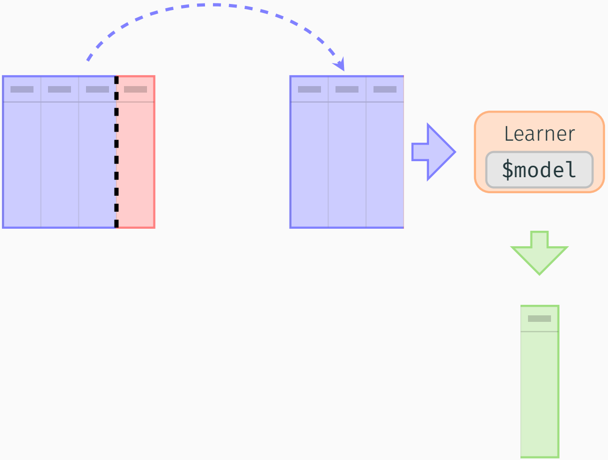




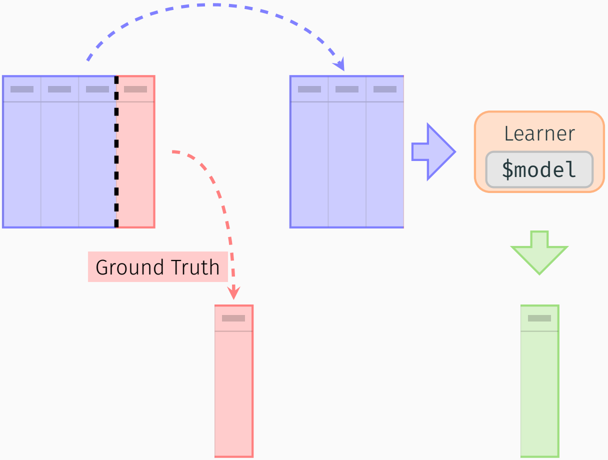
Performance Evaluation



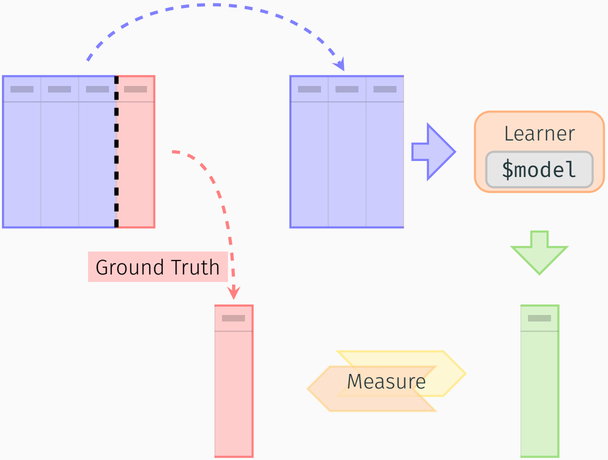
Performance Evaluation



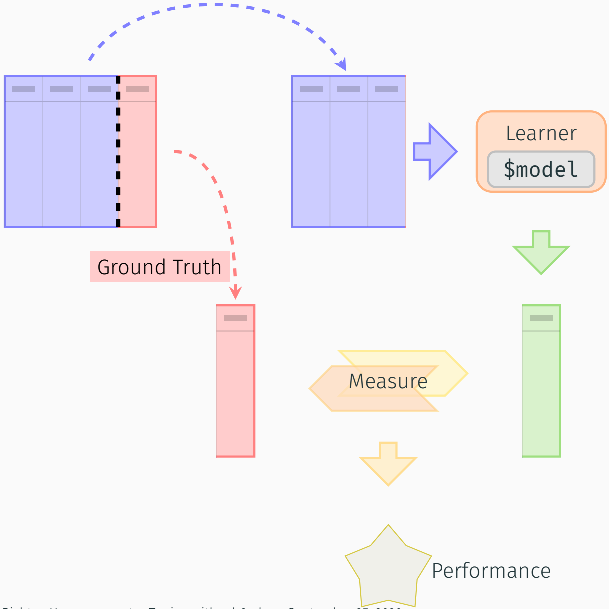
Performance Evaluation



Performance Evaluation



Performance Evaluation



Performance Evaluation

- Prediction ‘Task‘ with known data

```
known_truth_task$data()
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

Performance Evaluation

- Prediction ‘Task‘ with known data

```
known_truth_task$data()
```

```
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred
```

```
## <PredictionClassif> for 2 observations:
##  row_id  truth  response
##       1 setosa    setosa
##       2 setosa virginica
```

Performance Evaluation

- Prediction ‘Task‘ with known data

```
known_truth_task$data()
```

```
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred
```

```
## <PredictionClassif> for 2 observations:
##  row_id  truth  response
##       1 setosa    setosa
##       2 setosa virginica
```

- Score the prediction

```
pred$score(msr("classif.ce"))
```

```
## classif.ce
##        0.5
```

Performance Evaluation

- Prediction 'Task' with known data

```
known_truth_task$data()
```

```
#    Species Petal.Length Petal.Width Sepal.Length Sepal.Width
#     <fctr>        <num>       <num>        <num>       <num>
# 1:  setosa            2           1            4           3
# 2:  setosa            3           2            2           2
```

- Predict again

```
pred = learner$predict(known_truth_task)
pred
```

```
## <PredictionClassif> for 2 observations:
##  row_id  truth  response
##       1 setosa    setosa
##       2 setosa virginica
```

- Score the prediction

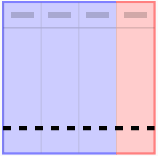
```
pred$score(msr("classif.ce"))
```

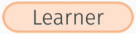
```
## classif.ce
##        0.5
```

Resampling

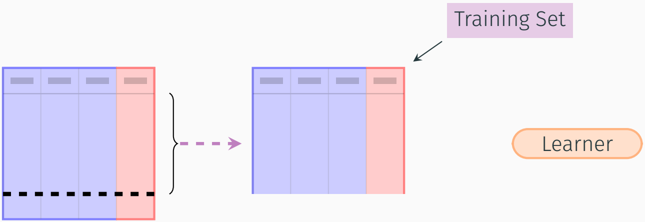
Resampling

Resampling

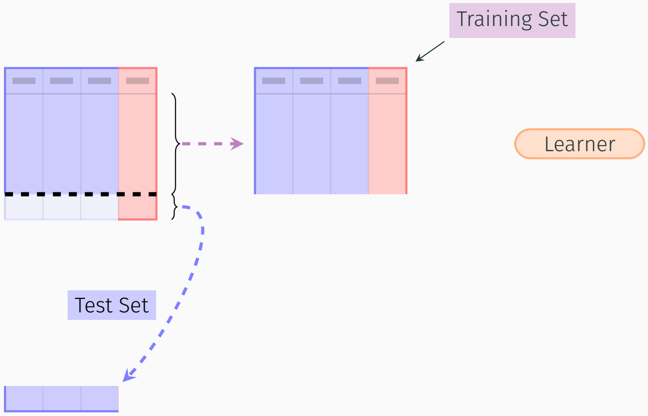




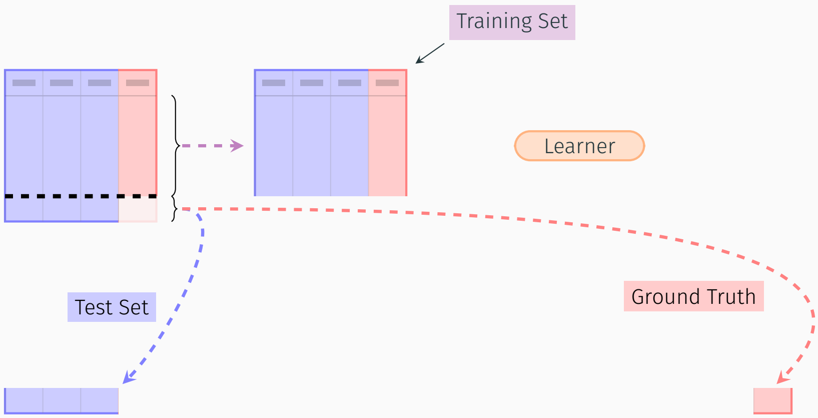
Resampling



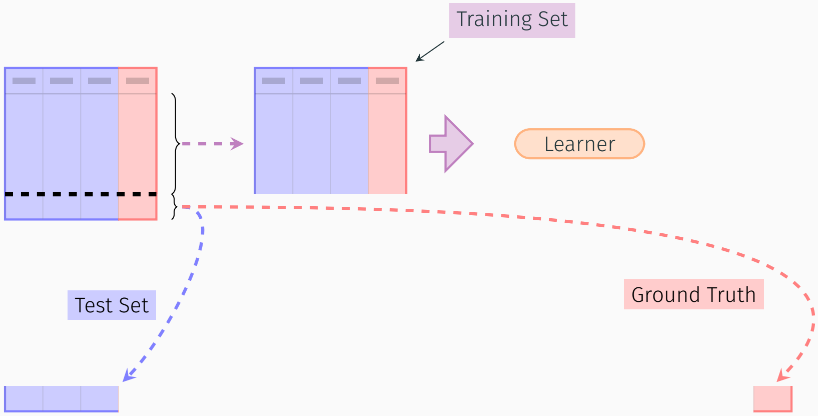
Resampling



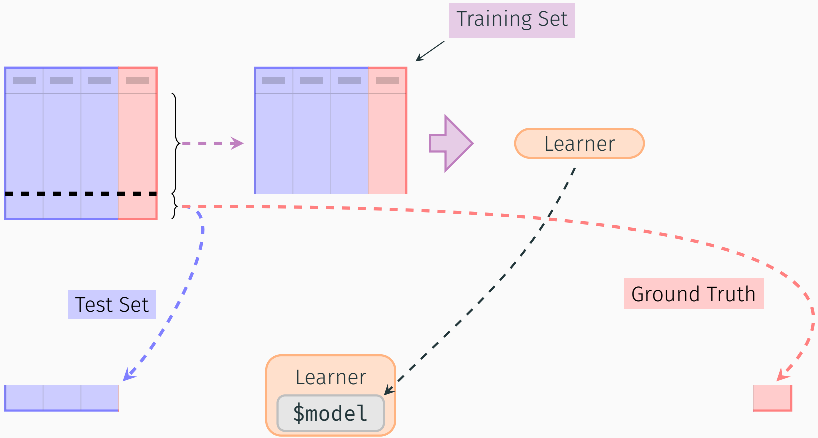
Resampling



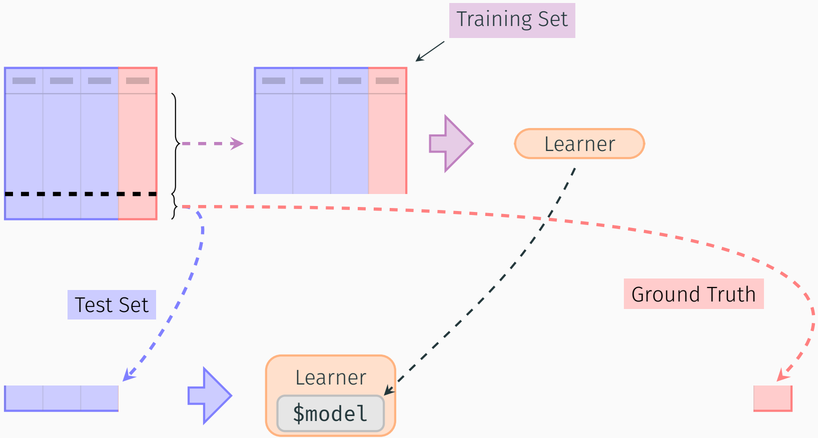
Resampling



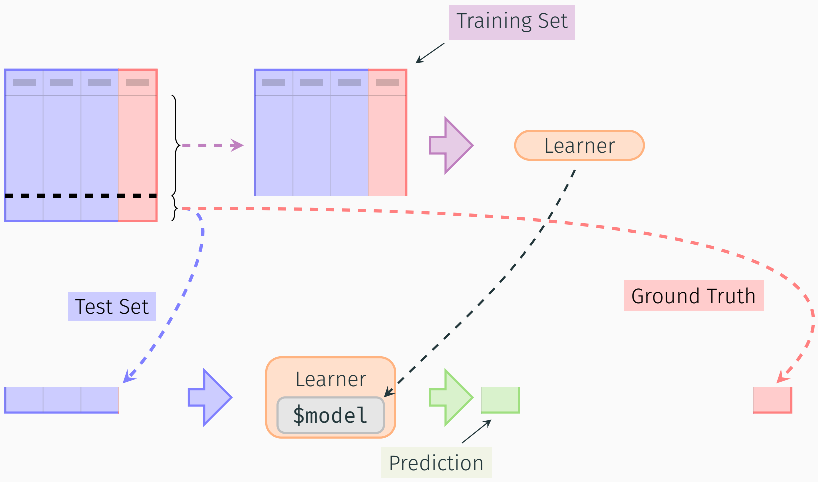
Resampling



Resampling

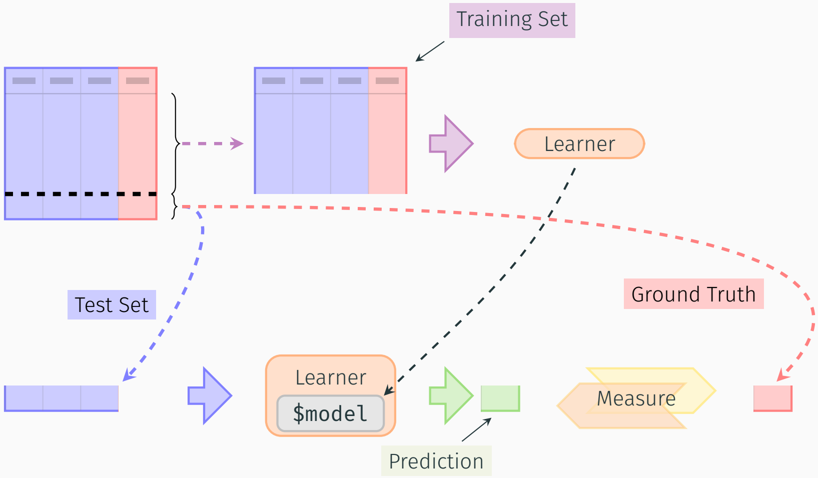


Resampling

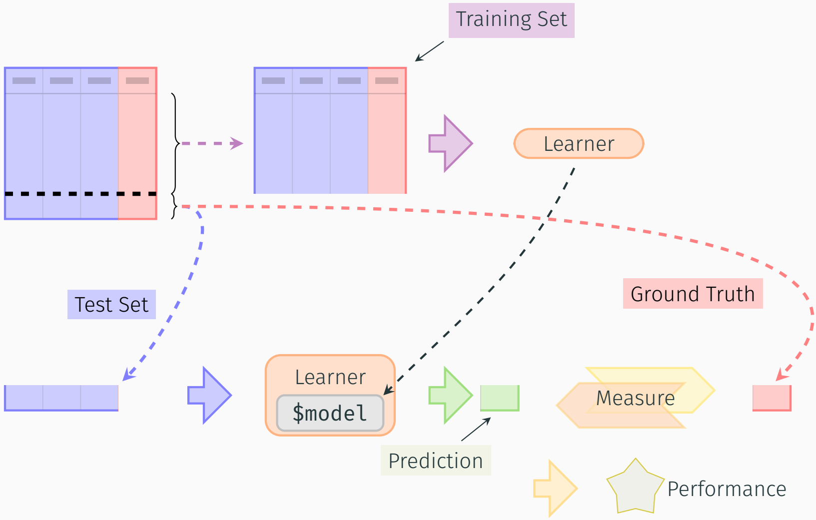


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Prediction

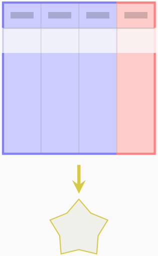
Resampling


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction

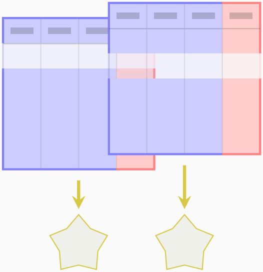
Resampling


Training Set
Learner
Test Set
Ground Truth
Learner
$model
Measure
Prediction
Performance

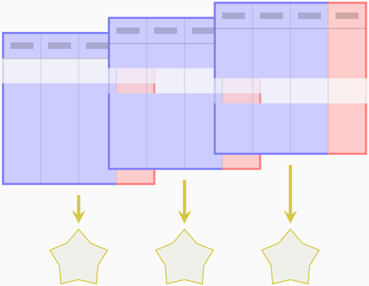
Resampling



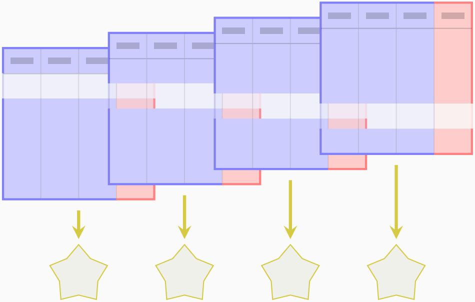
Resampling



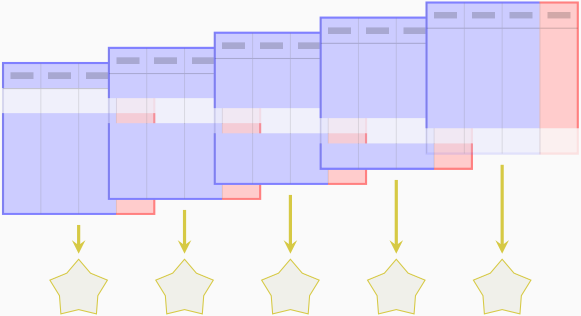
Resampling



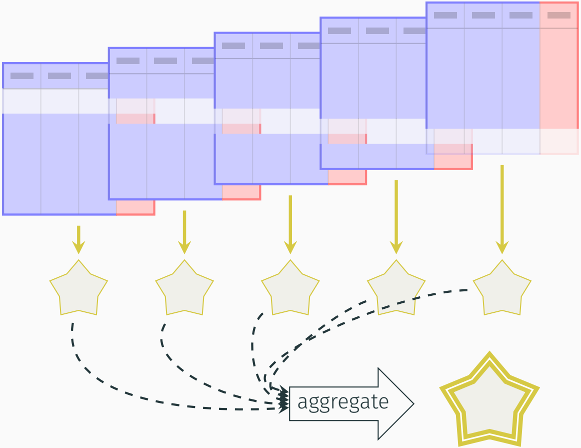
Resampling



Resampling

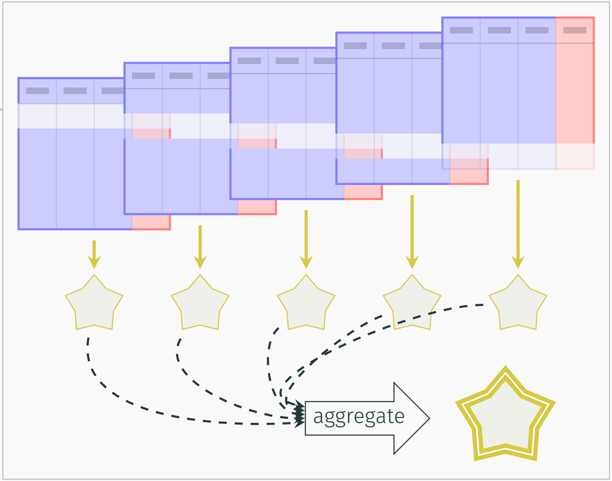


Resampling



Resampling

`resample()`



Resampling

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

Resampling

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
rr = resample(task, learner, cv5)
```

Resampling

- Resample description: How to split the data

```
cv5 = rsmp("cv", folds = 5)
```

- Use the `resample()` function for resampling:

```
rr = resample(task, learner, cv5)
```

- We get a `ResamplingResult` object:

```
print(rr)

## <ResampleResult> of 5 iterations
## * Task: iris
## * Learner: classif.rpart
## * Warnings: 0 in 0 iterations
## * Errors: 0 in 0 iterations
```

What exactly is a `ResamplingResult` object?

Resampling Results

What exactly is a `ResamplingResult` object?

Remember `Prediction`:

Resampling Results

What exactly is a `ResamplingResult` object?

Remember `Prediction`:

- Get a table representation using `as.data.table()`

```
rr_table = as.data.table(rr)

print(rr_table)
#                 task                   learner        resampling
#               <list>                    <list>            <list>
# 1: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 2: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 3: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 4: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 5: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
#    iteration              prediction
#        <int>                  <list>
# 1:         1 <PredictionClassif[19]>
# 2:         2 <PredictionClassif[19]>
# 3:         3 <PredictionClassif[19]>
# 4:         4 <PredictionClassif[19]>
# 5:         5 <PredictionClassif[19]>
```

Resampling Results

What exactly is a `ResamplingResult` object?

Remember `Prediction`:

- Get a table representation using `as.data.table()`

```
rr_table = as.data.table(rr)

print(rr_table)
#                  task                      learner         resampling
#                <list>                       <list>             <list>
# 1: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 2: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 3: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 4: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
# 5: <TaskClassif[45]> <LearnerClassifRpart[32]> <ResamplingCV[19]>
#    iteration            prediction
#        <int>                <list>
# 1:         1 <PredictionClassif[19]>
# 2:         2 <PredictionClassif[19]>
# 3:         3 <PredictionClassif[19]>
# 4:         4 <PredictionClassif[19]>
# 5:         5 <PredictionClassif[19]>
```

- Active bindings and functions that make information easily accessible

Resampling Results

- Calculate performance:

```
rr$aggregate(msr("classif.ce"))
```

```
## classif.ce
##       0.06
```

- Calculate performance:

```
rr$aggregate(msr("classif.ce"))
## classif.ce
##       0.06
```

- Get predictions

```
rr$prediction()
## <PredictionClassif> for 150 observations:
##     row_id     truth  response
##          2    setosa    setosa
##         15    setosa    setosa
##         17    setosa    setosa
## ---
##        142 virginica virginica
##        143 virginica virginica
##        148 virginica virginica
```

Resampling

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]
```

```
## <PredictionClassif> for 30 observations:
##     row_id     truth  response
##          2    setosa    setosa
##         15    setosa    setosa
##         17    setosa    setosa
## ---
##        141 virginica virginica
##        145 virginica virginica
##        149 virginica virginica
```

Resampling

- Predictions of individual folds

```
predictions = rr$predictions()
predictions[[1]]
```

```
## <PredictionClassif> for 30 observations:
##     row_id     truth  response
##          2    setosa    setosa
##         15    setosa    setosa
##         17    setosa    setosa
## ---
##        141 virginica virginica
##        145 virginica virginica
##        149 virginica virginica
```

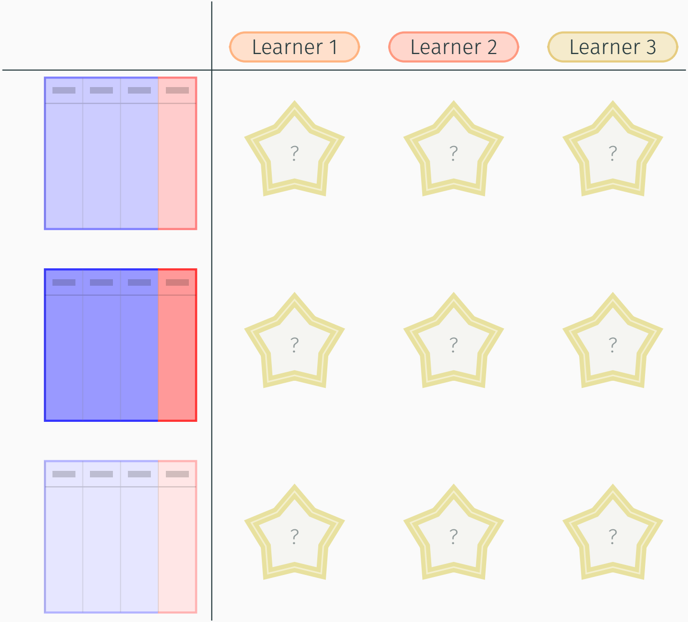
- Score of individual folds

```
scores = rr$score()
scores[1:3, c("iteration", "classif.ce")]
```

```
##    iteration classif.ce
##        <int>      <num>
## 1:         1      0.033
## 2:         2      0.100
## 3:         3      0.033
```

Benchmark

Performance Comparison



Performance Comparison

- Multiple Learners, multiple Tasks:

```r
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

Performance Comparison

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

Performance Comparison

- Multiple Learners, multiple Tasks:

```
library("mlr3learners")
learners = list(lrn("classif.rpart"), lrn("classif.kknn"))
tasks = list(tsk("iris"), tsk("sonar"), tsk("wine"))
```

- Set up the *design* and execute benchmark:

```
design = benchmark_grid(tasks, learners, cv5)
bmr = benchmark(design)
```

- We get a `BenchmarkResult` object which shows that `kknn` outperforms `rpart`:

```
bmr_ag = bmr$aggregate()
bmr_ag[, c("task_id", "learner_id", "classif.ce")]

##    task_id    learner_id classif.ce
##     <char>        <char>      <num>
## 1:    iris classif.rpart      0.047
## 2:    iris  classif.kknn      0.047
## 3:   sonar classif.rpart      0.269
## 4:   sonar  classif.kknn      0.144
## 5:    wine classif.rpart      0.112
## 6:    wine  classif.kknn      0.050
```

Benchmark Result

What exactly is a `BenchmarkResult` object?

Benchmark Result

What exactly is a `BenchmarkResult` object?

Just like `Prediction` and `ResamplingResult`!

Benchmark Result

What exactly is a `BenchmarkResult` object?

Just like `Prediction` and `ResamplingResult`!

- Table representation using `as.data.table()`

Benchmark Result

What exactly is a `BenchmarkResult` object?

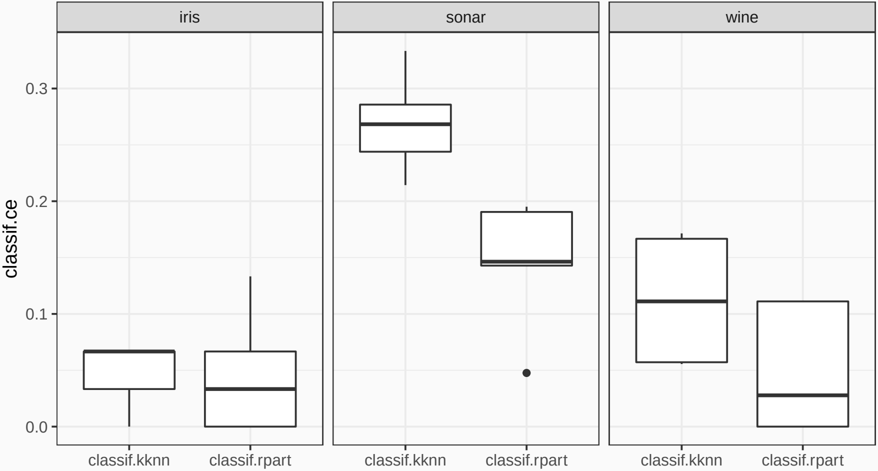
Just like `Prediction` and `ResamplingResult`!

- Table representation using `as.data.table()`
- Active bindings and functions that make information easily accessible

Benchmark Result

The `mlr3viz` package contains `autoplot()` functions for many mlr3 objects

```
library(mlr3viz)
autoplot(bmr)
```



Control of Execution

Control of Execution

Parallelization

```
future::plan("multicore")
```

- runs each resampling iteration as a job
- also allows nested resampling (although not needed here)

Encapsulation

```
learner$encapsulate = c(train = "callr", predict = "callr")
```

- Spawns a separate R process to train the learner
- Learner may segfault without tearing down the session
- Logs are captured
- Possibilty to have a fallback to create predictions

How to get Help

How to get Help

- Where to start?
    - Check these slides
    - Check the mlr3book https://mlr3book.mlr-org.com

How to get Help

- Where to start?
  - Check these slides
  - **Check the mlr3book https://mlr3book.mlr-org.com**
- Get help for R6 objects?
  1. Find out what kind of R6 object you have:

     ```
     class(bmr)
     ## [1] "BenchmarkResult" "R6"
     ```

  2. Go to the corresponding help page:

     ```
     ?BenchmarkResult
     ```

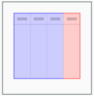
  New: open the corresponding man page with

```
learner$help()
```

mlr3 Outro

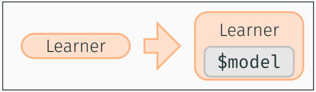
Ingredients:

Data



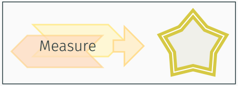
TaskClassif,
TaskRegr,
tsk()

Learning Algorithms



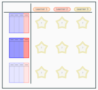
lrn() ⇒ Learner,
$train(),
$predict() ⇒ Prediction

Performance Evaluation



rsmp() ⇒ Resampling,
msr() ⇒ Measure,
resample() ⇒ ResamplingResult,
$aggregate()

Performance Comparison



benchmark_grid(),
benchmark() ⇒ BenchmarkResult

Black-Box Problems

Mathematical Problem Formulation

Optimization Problem:

$$y = f(\mathrm{x})\ , \quad f : \mathcal{X} \to \mathbb{R}$$
$$\mathrm{x}^* := \mathrm{argmin}_{\mathrm{x} \in \mathcal{X}} f(\mathrm{x})$$

But:

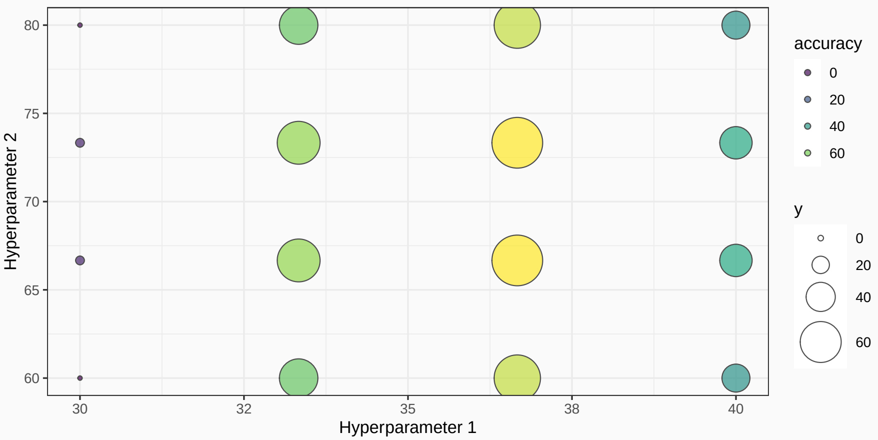
- $f'(\mathrm{x})$ unknown.
- $y = f(\mathrm{x}) + \varepsilon(\mathrm{x})$.
- Evaluation of $f(\mathrm{x})$ takes > 30 mins.

Note: W.l.o.g. we consider minimization problems. Maximization of f is equivalent to minimizing $-f$.

"Trial-and-Error" based on expert knowledge

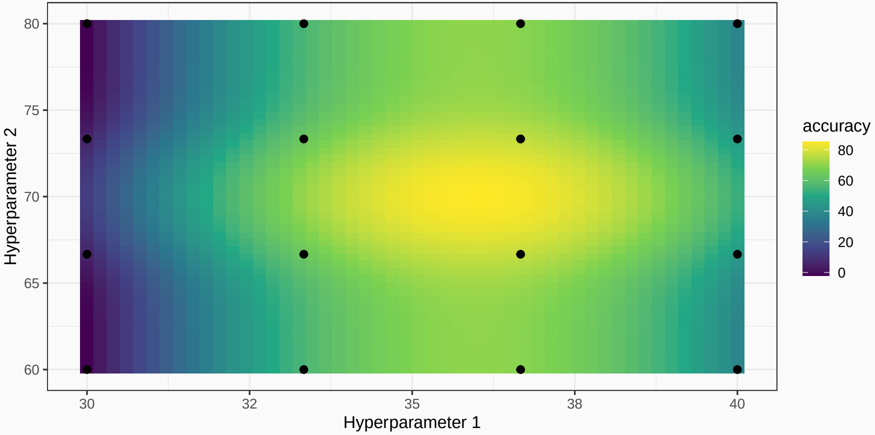
- ⊕ Can lead to fairly good outcomes for known problems
- ⊖ Very (!) inefficient
- ⊖ Poor reproducibility
- ⊖ Chosen solution can also be far away from a global optimum
- ⊖ What if there is no expert?

Grid search: Exhaustive search of a predefined grid of inputs

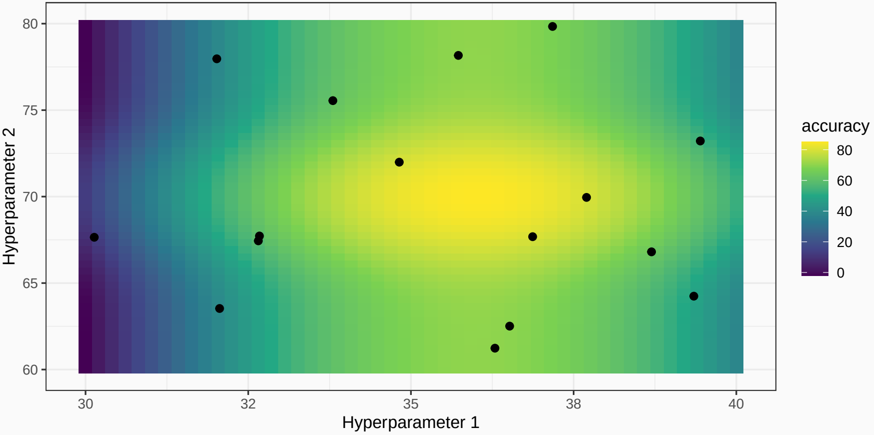


Naive Approaches II: Grid Search / Random Search / LHS

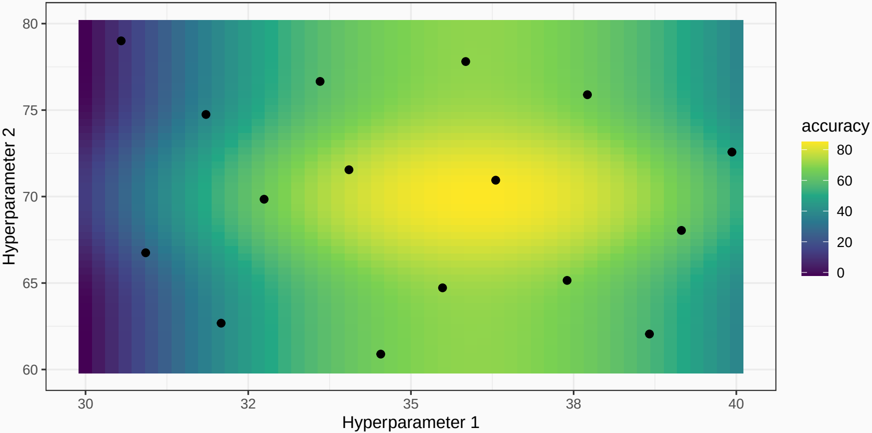
Grid search: Exhaustive search of a predefined grid of inputs



Random search: Evaluate uniformly sampled inputs



Latin hypercube sampling (LHS): inputs are sampled randomly, but no two inputs share the same value in a dimension.



Model-based Optimization

Basic MBO Idea

Black-Box: No additional information for f.

Only possibility: Selective evaluation of $f(\mathrm{x})$ and acquiring knowledge of evaluated points (x, y).

Wanted: Strategy to select x so that we get to the optimum quickly.

Basic MBO Idea

Black-Box: No additional information for f.

Only possibility: Selective evaluation of $f(\mathbf{x})$ and acquiring knowledge of evaluated points $(\mathbf{x}, y)$.

- Wanted: Strategy to select $\mathbf{x}$ so that we get to the optimum quickly.
- Idea: Evaluate $f(\mathbf{x})$ for some $\mathbf{x}$ and then fit a regression model $\hat{f}(\mathbf{x})$.

Basic MBO Idea

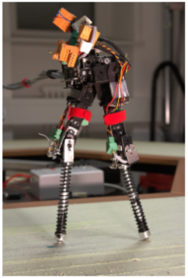
Black-Box: No additional information for f.

Only possibility: Selective evaluation of $f(\mathrm{x})$ and acquiring knowledge of evaluated points (x, y).

- Wanted: Strategy to select x so that we get to the optimum quickly.
- Idea: Evaluate $f(\mathrm{x})$ for some x and then fit a regression model $\hat{f}(\mathrm{x})$.
- Hope: Maximum of $\hat{f}(\mathrm{x})$ is close to maximum of $f(\mathrm{x})$.

Example Problem I: Robot Gait Optimization

Problems solved with MBO: Optimization the parametrized controller that steers robot's gait



- **Goal:** Find parameters s.t. velocity of the robot is maximized

Example Problem II: Optimizing a Cookie Recipe

Problems solved with MBO: Optimization of a cookie recipe

https://www.bettycrocker.com

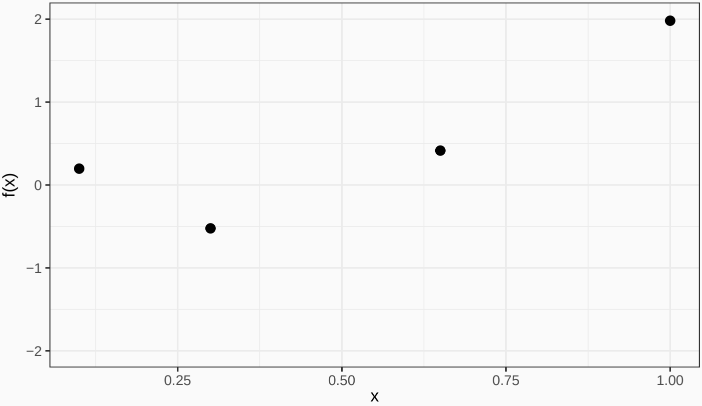
Ingredient	Salt (tsp)†	Total Sugar (g)	Brown Sugar (%)	Vanilla (tsp)†	Chip Quantity (g)	Chip Type
Min	0	150	0	0.25	114	{Dark, Milk, White}
Max	0.5	500	1	1	228	

- **Goal:** Find "optimal" amounts and composition of ingredients
- **Evaluation:** Cookies are baked according to the recipe, tested and rated by volunteers

Basic MBO Idea: Instructive Example

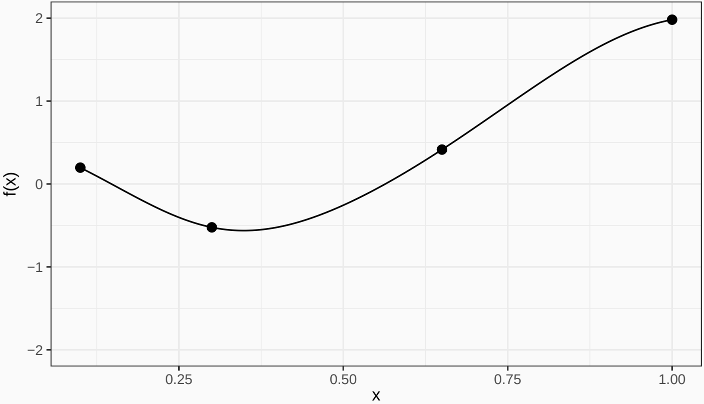
Starting Point:

- We evaluated f for a **some** inputs $\mathbf{x} \in \mathcal{X}$
- For now we assume that those evaluations are noise-free



Basic MBO Idea: Instructive Example

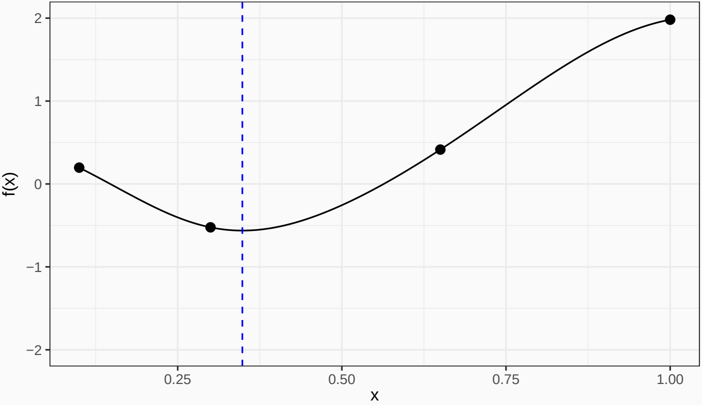
(i) **Fit** a **regression model** (black) to extract maximum information from the design points and learn properties of f



Note: As we can evaluate f without noise, we fit an interpolating regression model.

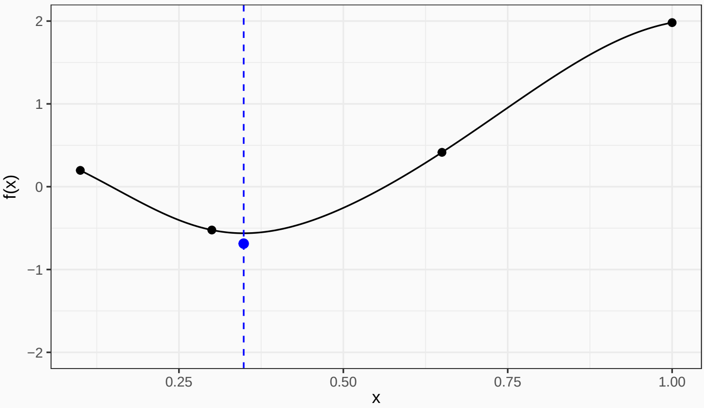
Basic MBO Idea: Instructive Example

(ii) Instead of the expensive f, we optimize the cheap model function (black) to **propose** a new point $\mathbf{x}^{\text{new}}$ for evaluation



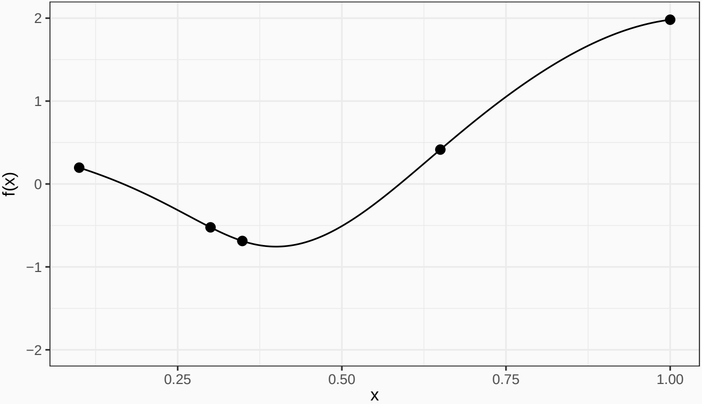
In the context of model-based optimization, the regression model is called **surrogate model**, because it is a cheap approximation of f that is iteratively trained.

(iii) And finally evaluate f on $\mathbf{x}^{(\text{new})}$



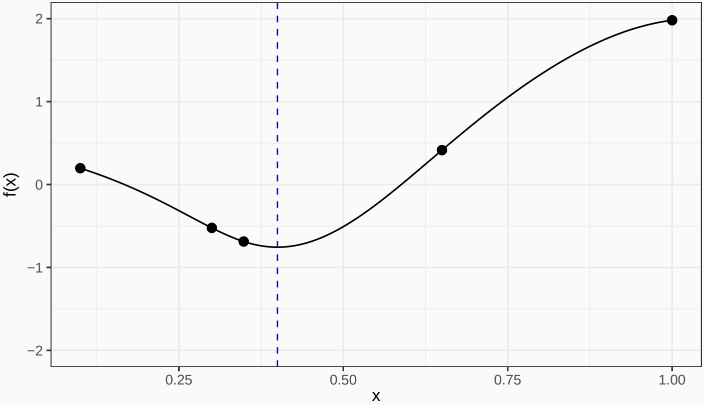
Basic MBO Idea: Instructive Example

After having evaluated the new point, we **adjust** the model on the expanded dataset via (slow) refitting or a (cheaper) online update



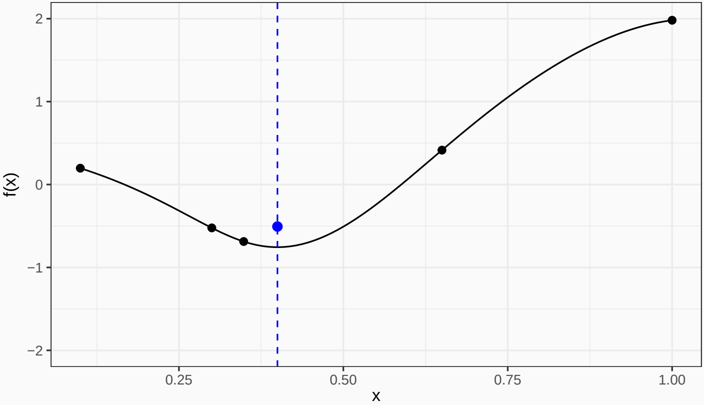
Basic MBO Idea: Instructive Example

We repeat: (i) **fit** the model, (ii) **propose** a new point and, (iii) **evaluate** that point.



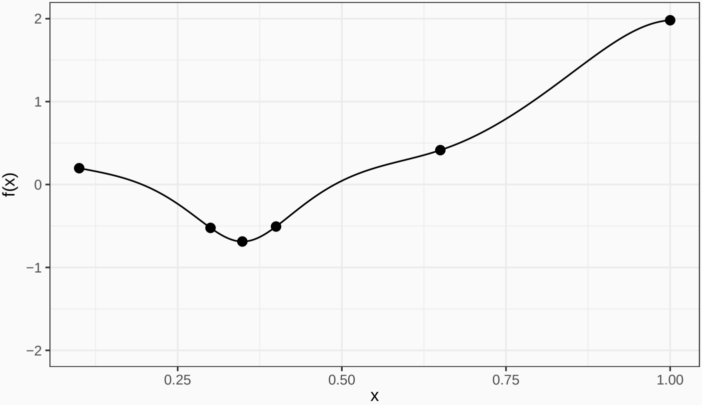
Basic MBO Idea: Instructive Example

We repeat: (i) **fit** the model, (ii) **propose** a new point and, (iii) **evaluate** that point.



Basic MBO Idea: Instructive Example

We repeat: (i) **fit** the model, (ii) **propose** a new point and, (iii) **evaluate** that point.



Basic MBO Idea: Intermediate Summary

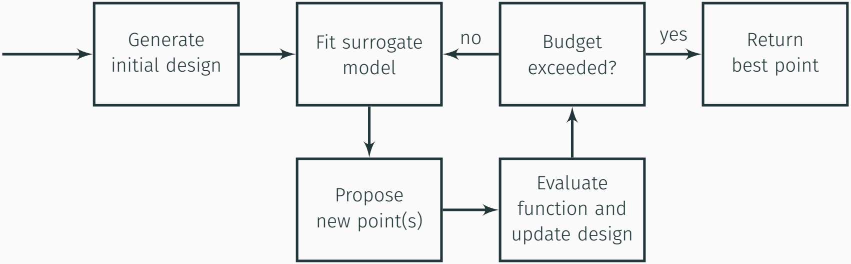
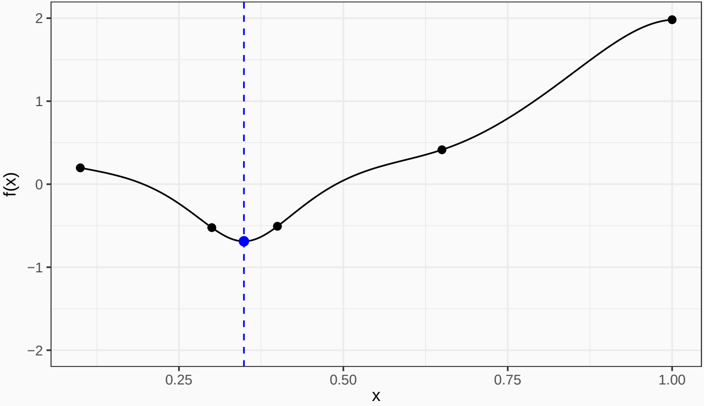


Figure 1: General SMBO approach.

Exploration-Exploitation-Tradeoff

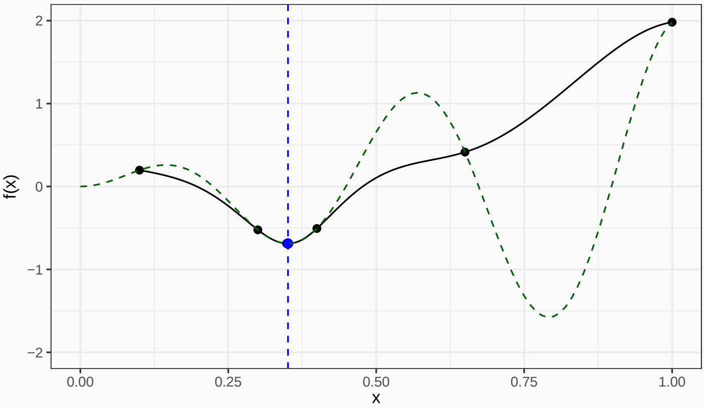
Exploration vs. Exploitation

In the example, the algorithm has converged. The sketched optimization procedure would return the point $x = 0.35$.



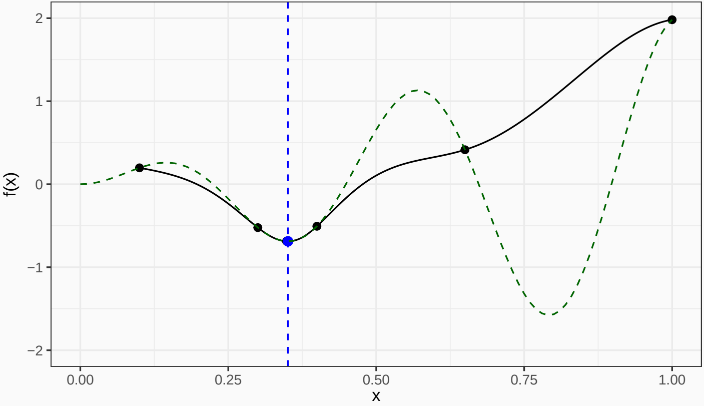
Exploration vs. Exploitation

The dashed green line is the “unknown” black-box function the sequential optimization procedure has been applied to.



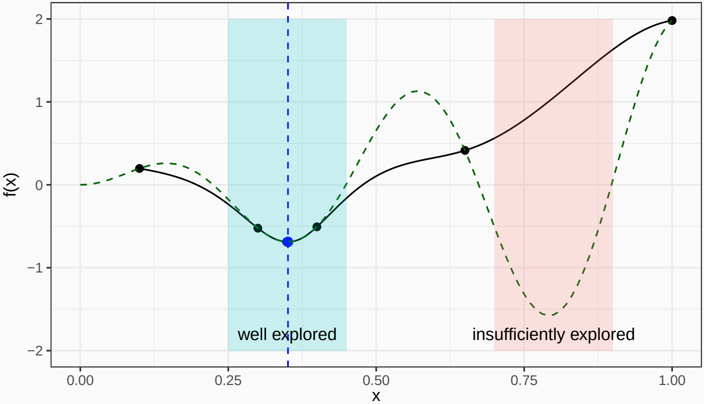
Exploration vs. Exploitation

We see: We ran into a local minimum. We did not “explore” the most crucial areas and **missed** the global minimum.



Goal:

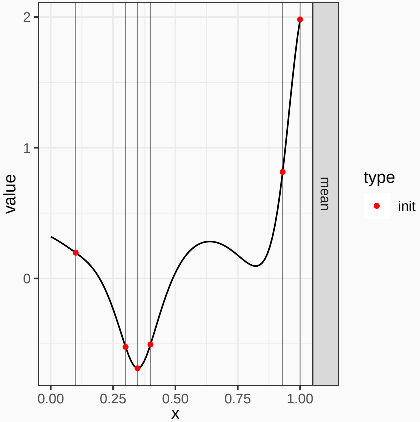
Find a trade-off between **exploration** (explore areas we do not know well) and **exploitation** (exploit interesting areas)



Exploration vs. Exploitation

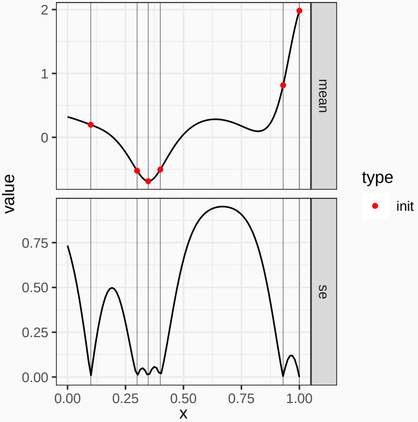
- How can we avoid under-exploration?

Basic MBO Idea: Example



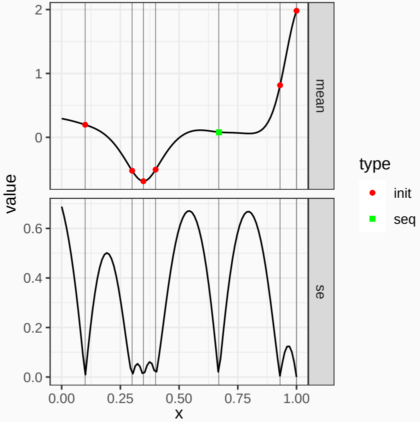
- Use regression method (e.g. a Gaussian Process) to get a prediction for different $\mathbf{x}$.
- Prediction of $\hat{f}(\mathbf{x})$ does not help, as its optimum has already been evaluated.

Basic MBO Idea: Example



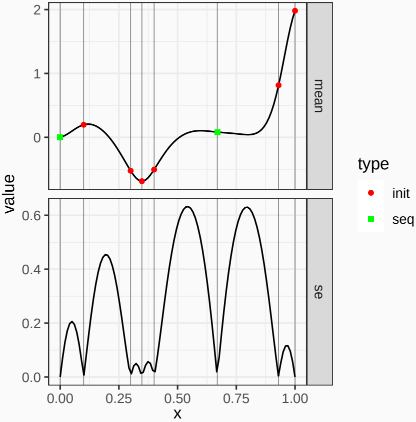
- Use regression method (e.g. a Gaussian Process) to get a prediction for different **x**.
- Prediction of $\hat{f}(\mathbf{x})$ does not help, as its optimum has already been evaluated.
- We need to explore: Use an **uncertainty** estimate $\hat{s}(\mathbf{x})$ to find uncertain regions.

Basic MBO Idea: Example



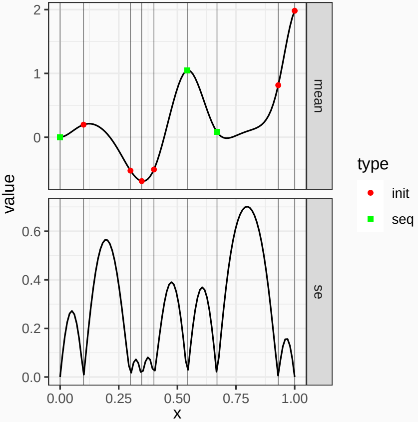
- Use regression method (e.g. a Gaussian Process) to get a prediction for different **x**.
- Prediction of $\hat{f}(\mathbf{x})$ does not help, as its optimum has already been evaluated.
- We need to explore: Use an **uncertainty** estimate $\hat{s}(\mathbf{x})$ to find uncertain regions.
- However: "Bad" areas with high uncertainty uninteresting.

Basic MBO Idea: Example



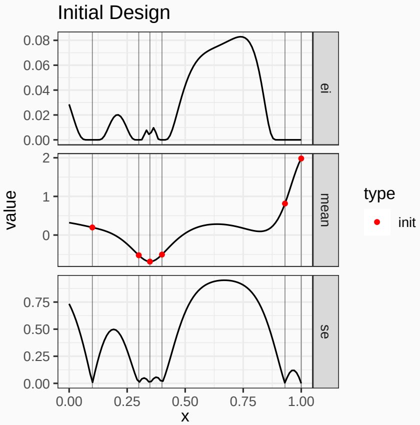
- Use regression method (e.g. a Gaussian Process) to get a prediction for different **x**.
- Prediction of $\hat{f}(\mathbf{x})$ does not help, as its optimum has already been evaluated.
- We need to explore: Use an **uncertainty** estimate $\hat{s}(\mathbf{x})$ to find uncertain regions.
- However: "Bad" areas with high uncertainty uninteresting.

Basic MBO Idea: Example



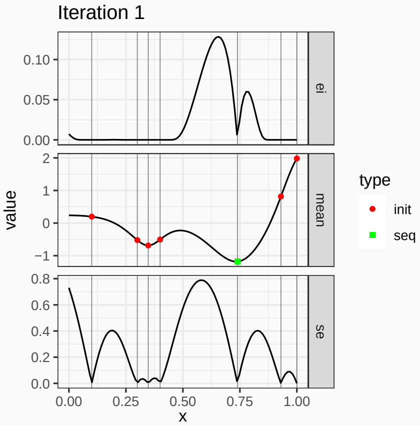
- Use regression method (e.g. a Gaussian Process) to get a prediction for different **x**.
- Prediction of $\hat{f}(\mathbf{x})$ does not help, as its optimum has already been evaluated.
- We need to explore: Use an **uncertainty** estimate $\hat{s}(\mathbf{x})$ to find uncertain regions.
- However: "Bad" areas with high uncertainty uninteresting.

Basic MBO Idea: Example



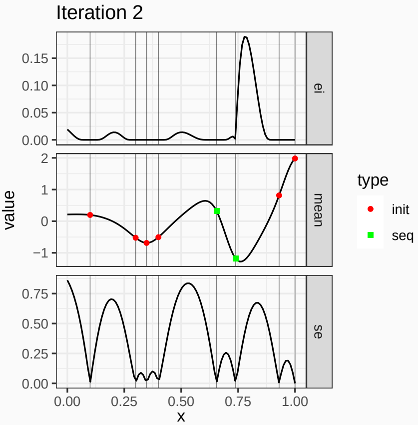
💡 **Idea**: Combine mean prediction and uncertainty via an **infill criterion**, that tells us where to search next.

Basic MBO Idea: Example



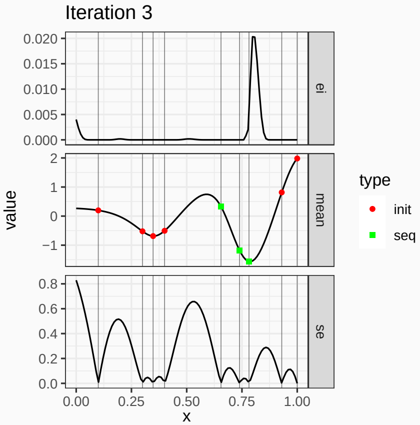
- **Idea**: Combine mean prediction and uncertainty via an **infill criterion**, that tells us where to search next.
  - The most prominent infill criterion is the **expected improvement** $EI(\mathbf{x}) = \mathbb{E}(I(\mathbf{x}))$ with $I(\mathbf{x}) := \max\left\{\hat{f}(\mathbf{x}) - y_{\min}, 0\right\}$

Basic MBO Idea: Example



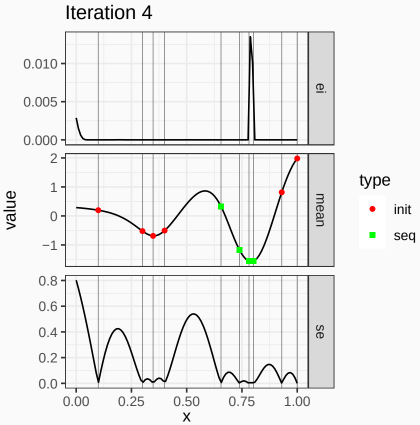
- **Idea**: Combine mean prediction and uncertainty via an **infill criterion**, that tells us where to search next.
- The most prominent infill criterion is the **expected improvement** $EI(\mathbf{x}) = \mathbb{E}(I(\mathbf{x}))$ with $I(\mathbf{x}) := \max\left\{\hat{f}(\mathbf{x}) - y_{\min}, 0\right\}$

Basic MBO Idea: Example



- **Idea**: Combine mean prediction and uncertainty via an **infill criterion**, that tells us where to search next.
- The most prominent infill criterion is the **expected improvement** $EI(\mathbf{x}) = \mathbb{E}(I(\mathbf{x}))$ with $I(\mathbf{x}) := \max\left\{\hat{f}(\mathbf{x}) - y_{\min}, 0\right\}$

Basic MBO Idea: Example



- **Idea**: Combine mean prediction and uncertainty via an **infill criterion**, that tells us where to search next.
- The most prominent infill criterion is the **expected improvement** $EI(\mathbf{x}) = \mathbb{E}(I(\mathbf{x}))$ with $I(\mathbf{x}) := \max\left\{\hat{f}(\mathbf{x}) - y_{\min}, 0\right\}$

Basic MBO Idea: Summary

- Based on observed data D, fit a regression model (e.g. Kriging) that gives us
  1. a posterior mean $\hat{\mu}(\mathbf{x})$ (which was our model prediction before) and
  2. a posterior variance $\hat{s}(\mathbf{x})$ (model uncertainty)

  for unknown $\mathbf{x}$.
- Combine mean prediction and uncertainty via an **acquisition function**, that tells us where to search next.

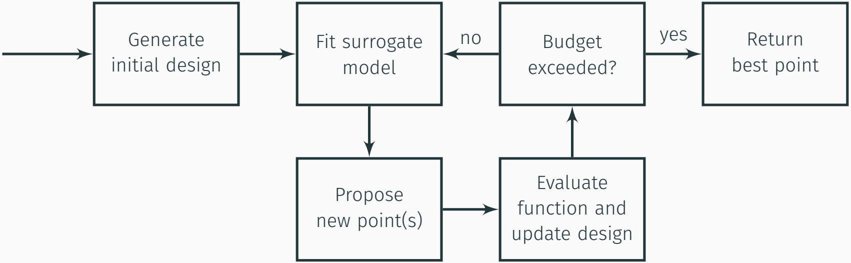


Figure 2: General SMBO approach.

Note: Model-based optimization is also called **Bayesian Optimization**.

Basic MBO Idea: Summary

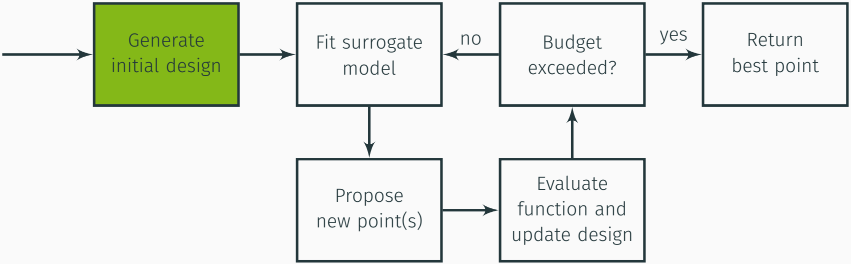
The user has three basic choices to make:

- What is the **initial design**?
- Which model should be used as **surrogate model**?
- What is the **acquisition function**? I.e. how should posterior mean and posterior variance be "weighted"?

We will discuss the three choices in the following.

Initial Design

Initial Design



Initial Design

- The initial design $\mathcal{D} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1,\dots,m_{\text{init}}}$ is used to train the **first** regression model.
- Input space should be covered sufficiently; commonly used designs:
  - Latin hypercube sampling (LHS)
  - Maximin designs (Minimum distance between points is maximized)

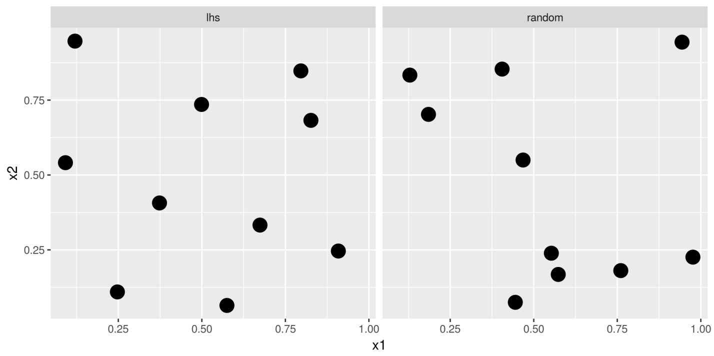


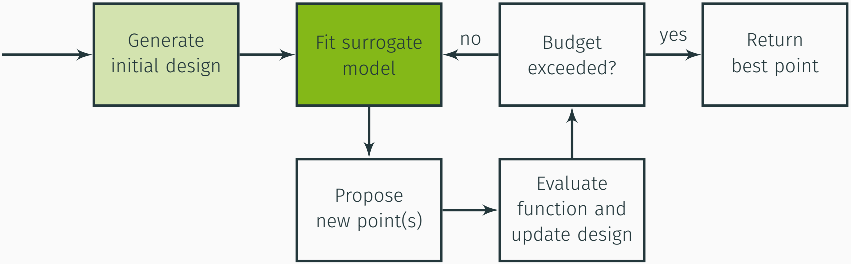
Figure 3: Latin hypercube design (left) vs. a random design (right).

Initial Design

- Type of design usually has not the largest effect on MBO; unequal distances between points could even be beneficial
- More important: **size** of the initial design
  - Too small $\rightarrow$ bad initial fit
  - Too large $\rightarrow$ spending too much budget without doing “intelligent” optimization
- Recommendations are based on the dimension of the input space d: $2d, 4d, 10d$

Surrogate Models

Surrogate Models



In general, any model that is capable of quantifying model uncertainty can be a suitable candidate.

We introduce two of the most commonly used surrogate models:

- Gaussian Processes
- Random Forests

A function $f(\mathrm{x})$ is generated by a GP $\mathcal{GP}\left(m(\mathrm{x}), k(\mathrm{x}, \mathrm{x}')\right)$ if for **any finite** set of inputs $\left\{\mathrm{x}^{(1)}, \dots, \mathrm{x}^{(n)}\right\}$, the associated vector of function values $\mathbf{f} = \left(f(\mathrm{x}^{(1)}), \dots, f(\mathrm{x}^{(n)})\right)$ has a Gaussian distribution

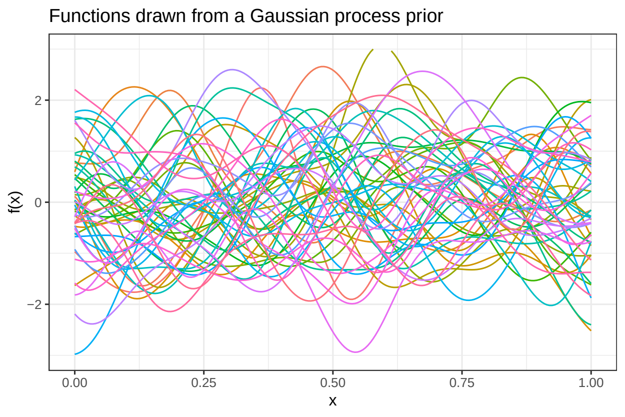
$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{m}, \mathbf{K}\right),$$

with

$$\mathbf{m} \quad := \quad \left(m\left(\mathrm{x}^{(i)}\right)\right)_i, \quad \mathbf{K} := \left(k\left(\mathrm{x}^{(i)}, \mathrm{x}^{(j)}\right)\right)_{i,j}.$$

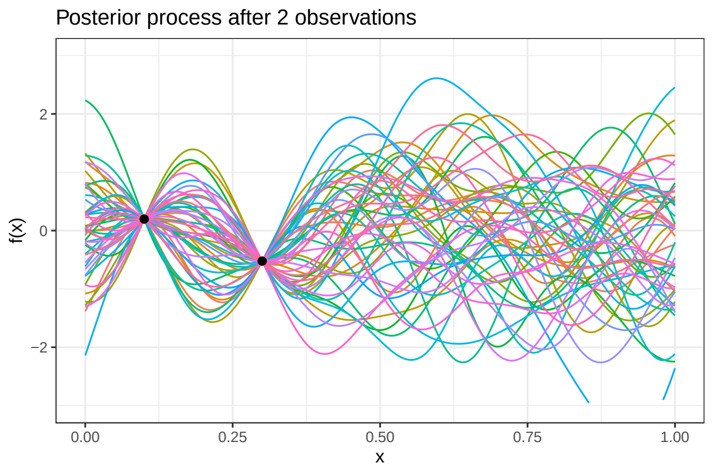
$m(\mathrm{x})$ is called mean function and $k(\mathrm{x}, \mathrm{x}')$ is called covariance function.

This way a **distribution over functions** is specified. It allows us to draw functions from this distribution.



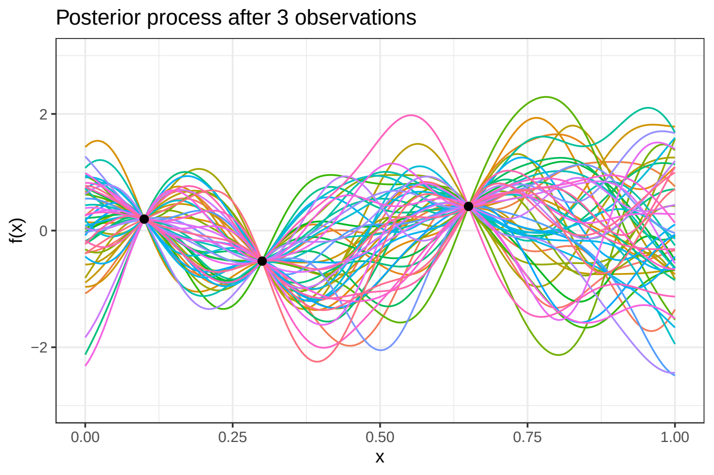
Gaussian Process Surrogate Model

After observing points, we consider some of the functions as “unrealistic”, because they are not consistent with the data. In turn, we get more and more certain about which functions are consistent with the data.

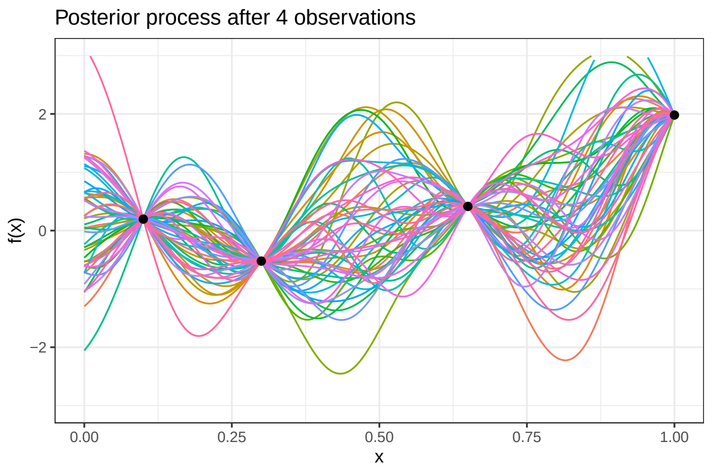


Gaussian Process Surrogate Model

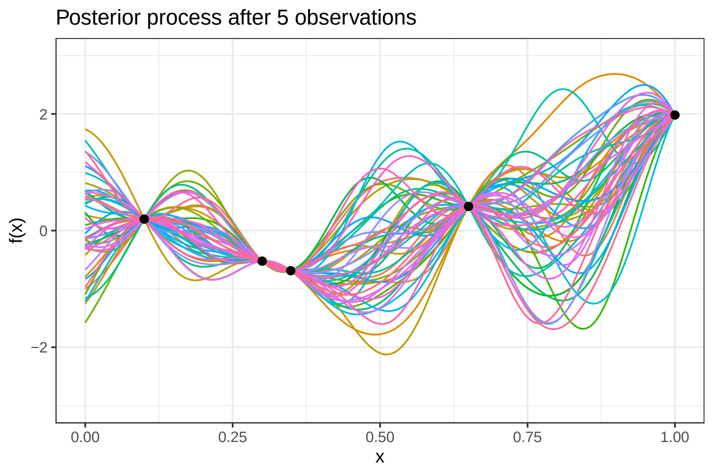
After observing points, we consider some of the functions as “unrealistic”, because they are not consistent with the data. In turn, we get more and more certain about which functions are consistent with the data.



After observing points, we consider some of the functions as "unrealistic", because they are not consistent with the data. In turn, we get more and more certain about which functions are consistent with the data.

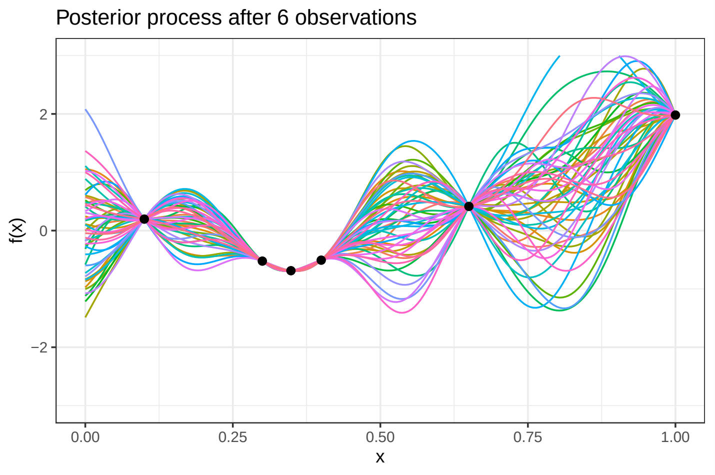


After observing points, we consider some of the functions as “unrealistic”, because they are not consistent with the data. In turn, we get more and more certain about which functions are consistent with the data.



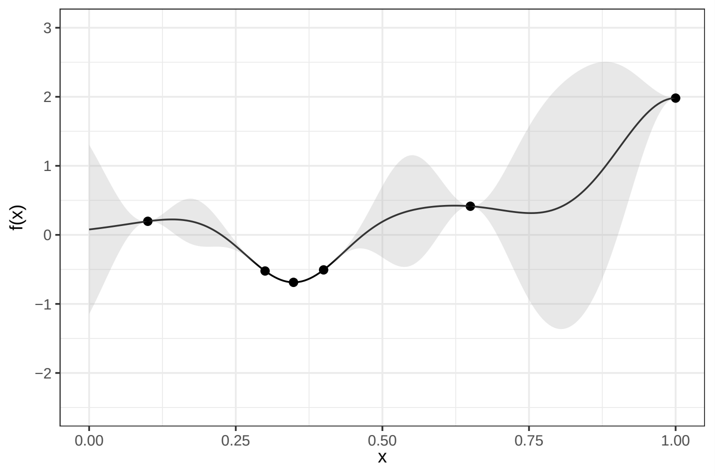
Gaussian Process Surrogate Model

After observing points, we consider some of the functions as “unrealistic”, because they are not consistent with the data. In turn, we get more and more certain about which functions are consistent with the data.



Gaussian Process Surrogate Model

The “variance” of the remaining functions are captured as model uncertainty.



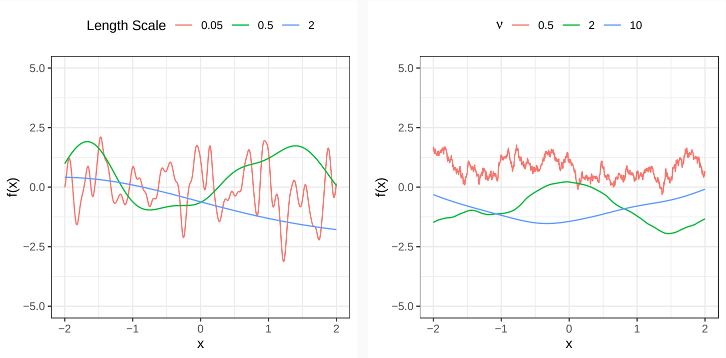
Covariance Function of a GP

Intuitively, the covariance function $k(\mathbf{x}, \mathbf{x}')$ is a **similarity** measure between points:

- if two points are close in $\mathcal{X}$, $k(\mathbf{x}, \mathbf{x}')$ is usually high - the correlation between the function values $f(\mathbf{x}), f(\mathbf{x}')$ is high
- if they are far away from each other, $k(\mathbf{x}, \mathbf{x}')$ is small and the function values are not correlated that strongly

The covariance function encodes **our assumptions** about the shape of the function we are interested in: how much variation in $f(\mathbf{x})$ we expect over which distances in $\mathcal{X}$.

Commonly used covariance functions

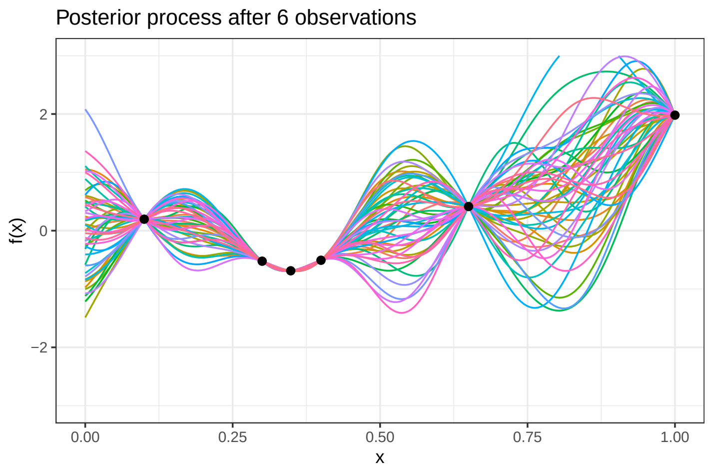


Random functions drawn from Gaussian processes with a Squared Exponential Kernel (left) and a Matern Kernel (right, $l = 1$).

The choice of the hyperparameter determines the "wiggliness" of the function.

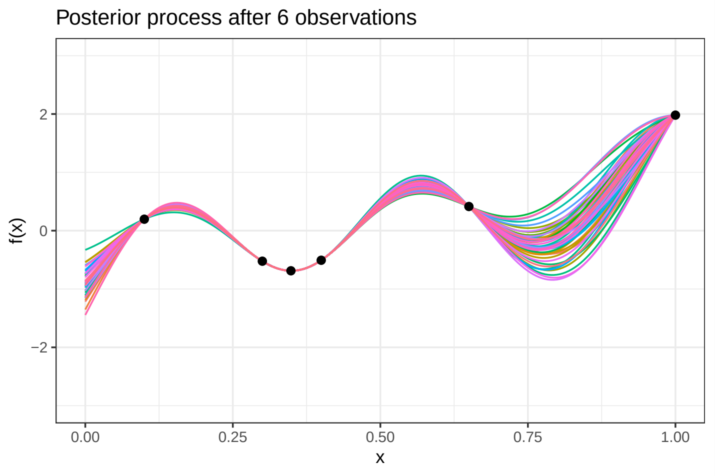
Kernel Parameters

The effect of the parametrization of the *Gaussian Process* on our previous example. Here with parameter $l = 0.1$.



Kernel Parameters

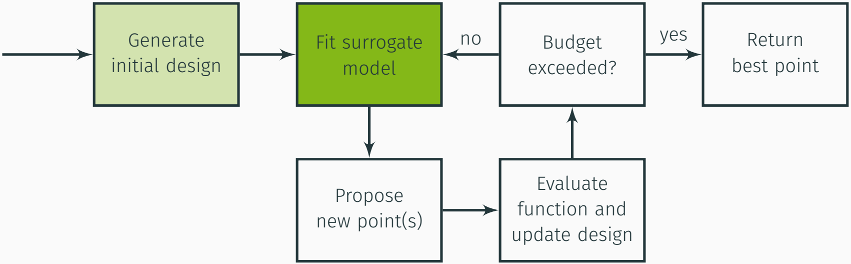
The effect of the parametrization of the *Gaussian Process* on our previous example. Here with parameter $l = 0.2$.



Gaussian Process Surrogate Model

- ⊕ Posterior variance is modeled as "spatial" uncertainty: Uncertainty increases with distance to design points, and is 0 directly at design points.
- ⊕ User can encode his assumptions about the shape of the function by specifying a covariance function.
- ⊖ Common kernels cannot represent discrete and hierarchical search spaces.

Surrogate Models: Random Forest



Problem: Kriging is not well suited for categorical search spaces

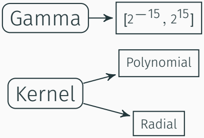


Figure 4: Example: Mixed search space

What is the distance between Kernel Radial and Polynomial?

Random Forest Surrogate Models ii

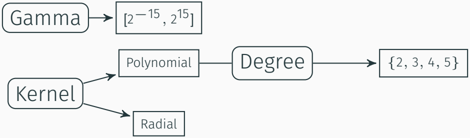


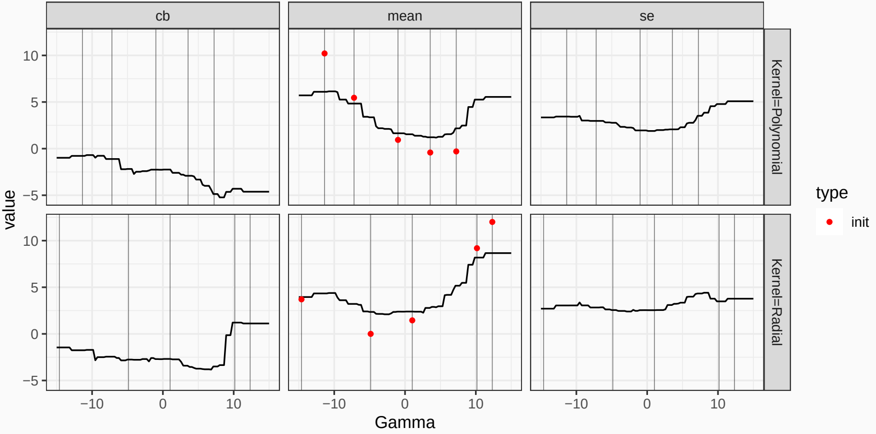
Figure 5: Example: Hierarchical search space

Ideas:

- Develop special distance measures.
- **Use regression model that is not distance based.**
  - $\Rightarrow$ Random Forest Regression
  - Needs handling of missing values.
  - Needs handling with factor variables.
  - Bad at learning interactions.

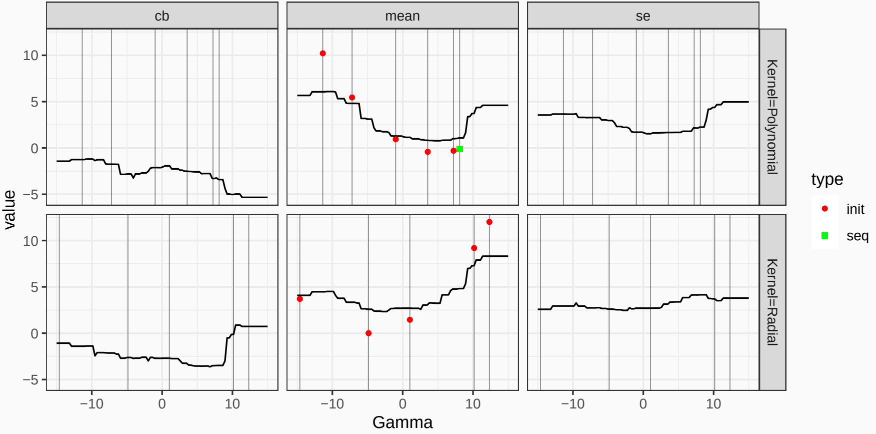
Random Forest Example

Random Forest is bad at learning interactions.



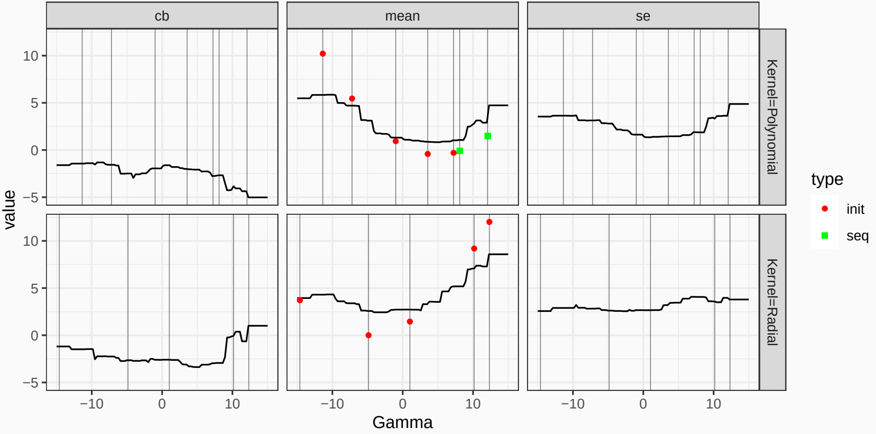
Random Forest Example

Random Forest is bad at learning interactions.



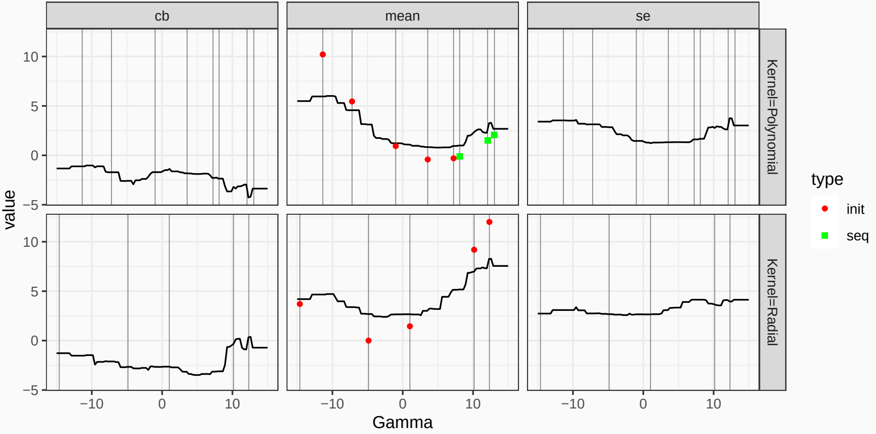
Random Forest Example

Random Forest is bad at learning interactions.



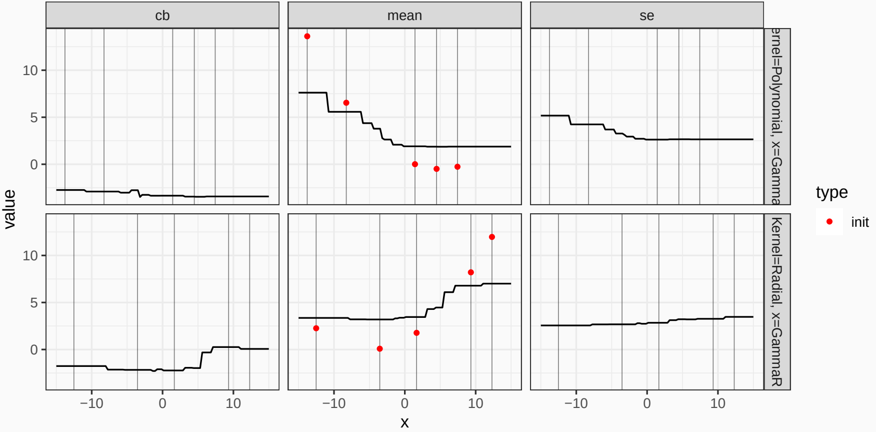
Random Forest Example

Random Forest is bad at learning interactions.



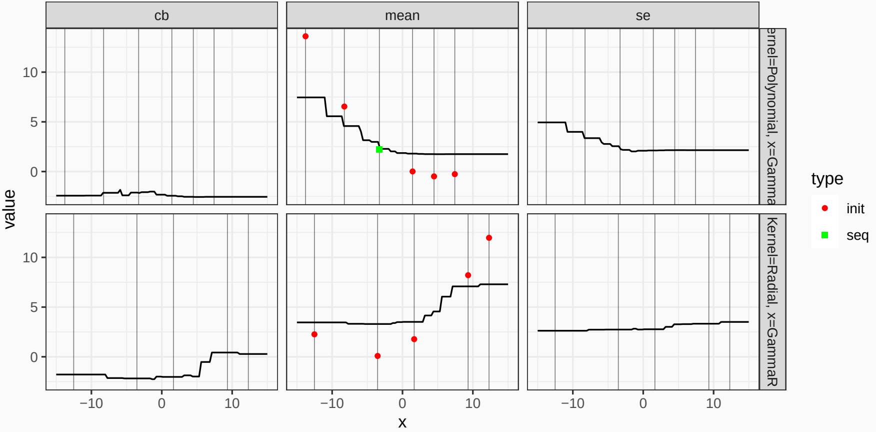
Random Forest Example

Solution: Treat parameter `Gamma` individually as `GammaR(adial)` and `GammaP(olynomial)`.



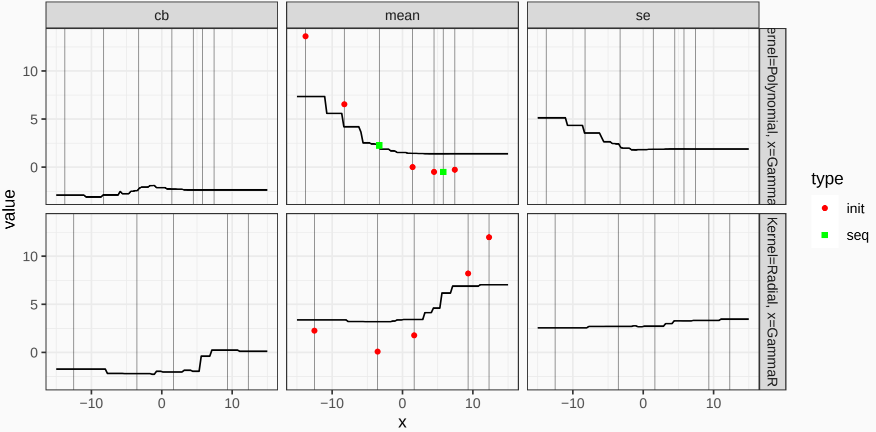
Random Forest Example

Solution: Treat parameter `Gamma` individually as `GammaR(adial)` and `GammaP(olynomial)`.



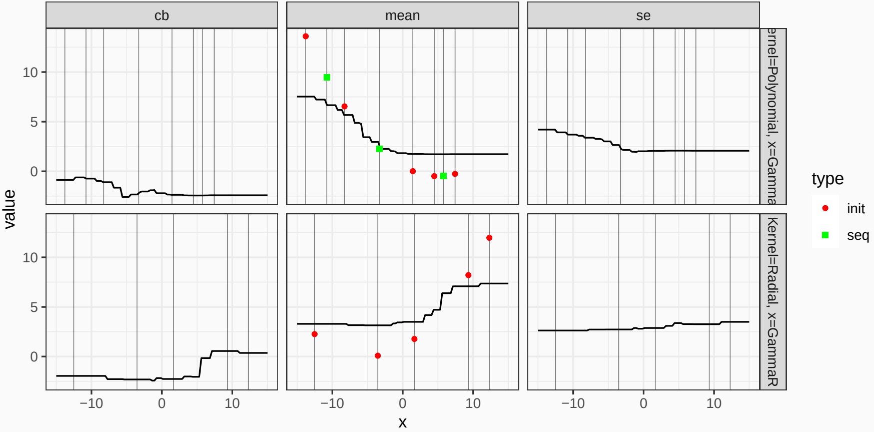
Random Forest Example

Solution: Treat parameter `Gamma` individually as `GammaR(adial)` and `GammaP(olynomial)`.



Random Forest Example

Solution: Treat parameter `Gamma` individually as `GammaR(adial)` and `GammaP(olynomial)`.



Benchmark Results [1]

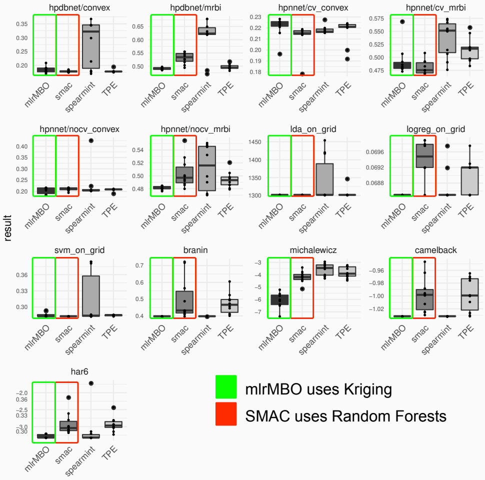


Figure 6: Benchmark results show that MBO optimizers with random forest surrogates perform badly on numerical problems.

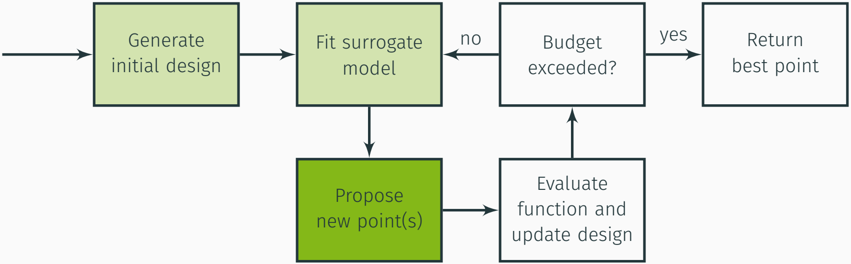
[1] https://arxiv.org/abs/1703.03373

Random Forest Characteristics

- ⊕ Handles categorical data.
- ⊕ Suited to represent hierarchical search spaces.
- ⊖ No extrapolation
  $\Rightarrow$ Initial design important!
- ⊖ Uncertainty only reflects signal variance.

Acquisition Functions

Acquisition Functions



Acquisition Functions

Recall: The acquisition function balances posterior mean and the posterior variance.

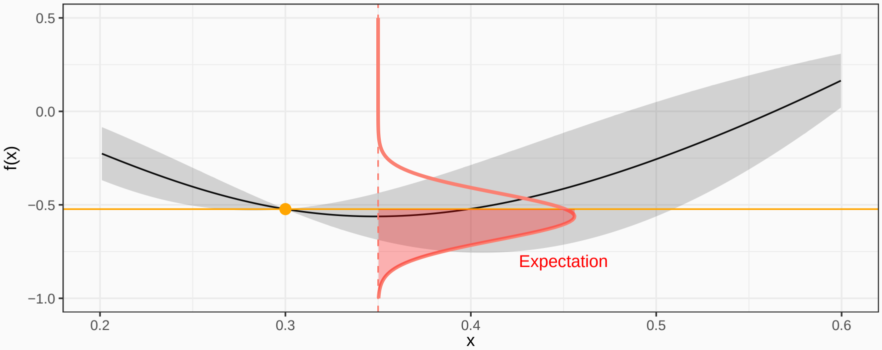
There are many different ways of balancing posterior mean and variance. We will discuss two of the most common ones:

- Expected Improvement
- Lower Confidence Bound

Goal: Propose $\mathbf{x}^{\text{new}}$ that maximizes the **expected improvement**:

$$EI(\mathbf{x}) \quad = \quad \mathbb{E}(\max\{y^{\min} - f(\mathbf{x}), 0\})$$

- Uncertainty only enters in the case of improvement $y^{\min} - f(\mathbf{x}) > 0$.



Expected Improvement

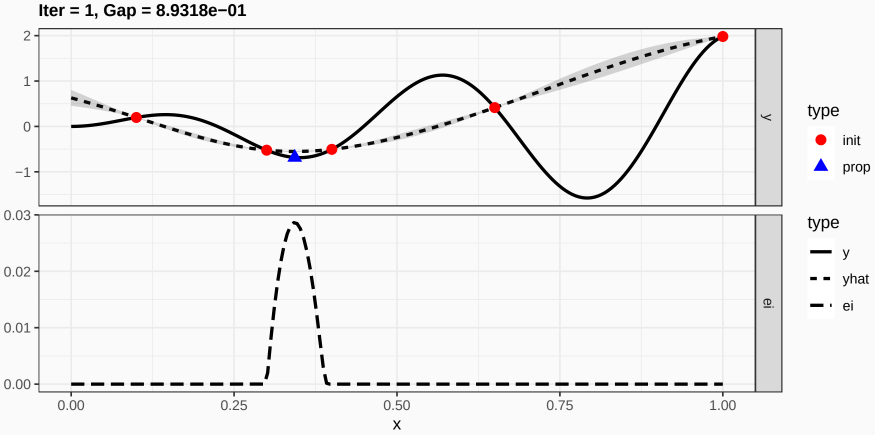
For a GP, i.e. $f(\mathrm{x}) \sim \mathcal{N}\left(\hat{\mu}(\mathrm{x}), \hat{s}^2(\mathrm{x})\right)$, we can express the $EI(\mathrm{x})$ in closed-form as:

$$EI(\mathrm{x}) = (y^{\min} - \hat{\mu}(\mathrm{x}))\Phi\Big(\frac{y^{\min} - \hat{\mu}(\mathrm{x})}{\hat{s}(\mathrm{x})}\Big) + \hat{s}(\mathrm{x})\phi\Big(\frac{y^{\min} - \hat{\mu}(\mathrm{x})}{\hat{s}(\mathrm{x})}\Big),$$

where $\phi(\cdot)$ denotes the density function of a standard normal random variable.

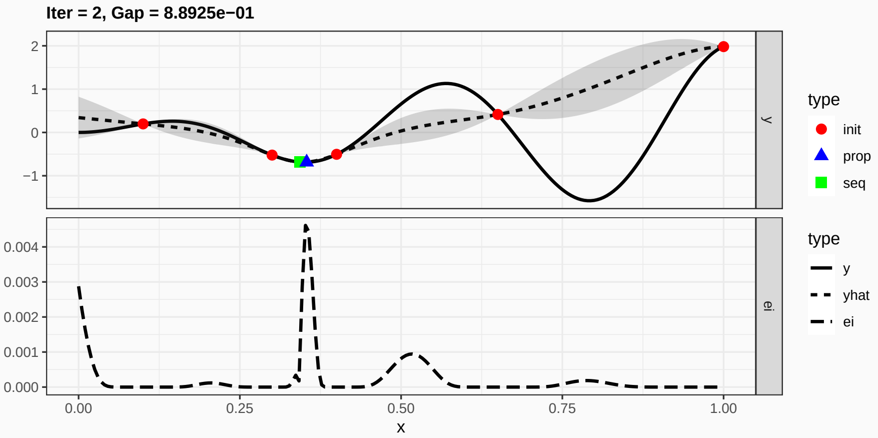
Expected Improvement

The expected improvement proposes a point that yields a bigger (potential) improvement than the point proposed by the probability of improvement.



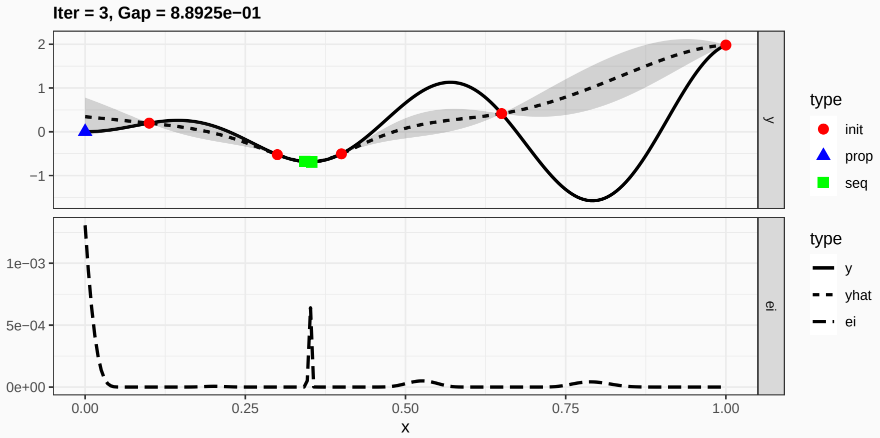
Expected Improvement

The expected improvement proposes a point that yields a bigger (potential) improvement than the point proposed by the probability of improvement.



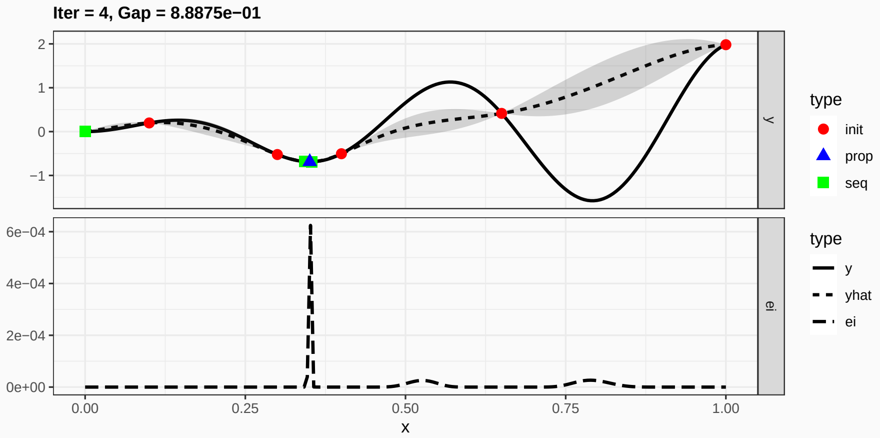
Expected Improvement

The expected improvement proposes a point that yields a bigger (potential) improvement than the point proposed by the probability of improvement.



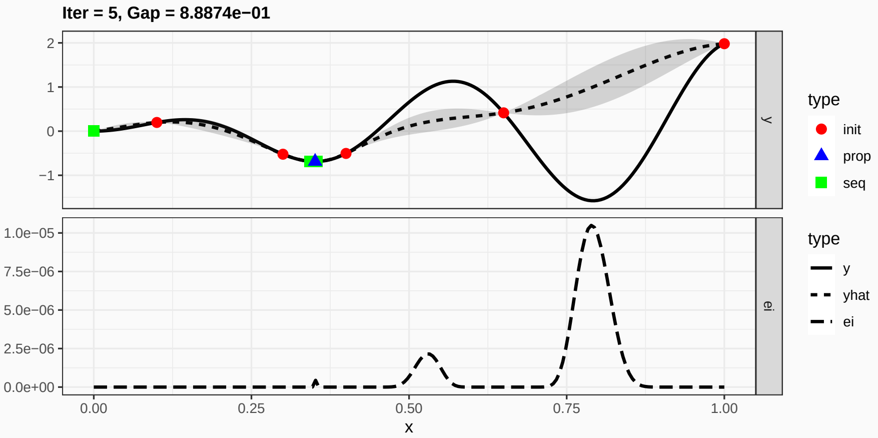
Expected Improvement

The expected improvement proposes a point that yields a bigger (potential) improvement than the point proposed by the probability of improvement.



Expected Improvement

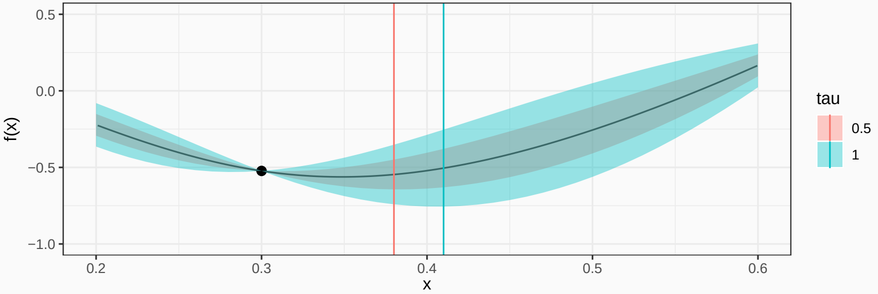
The expected improvement proposes a point that yields a bigger (potential) improvement than the point proposed by the probability of improvement.



Lower confidence bound (LCB):

$$LCB(\mathrm{x}) = \hat{\mu}(\mathrm{x}) - \tau \cdot \hat{s}(\mathrm{x}).$$

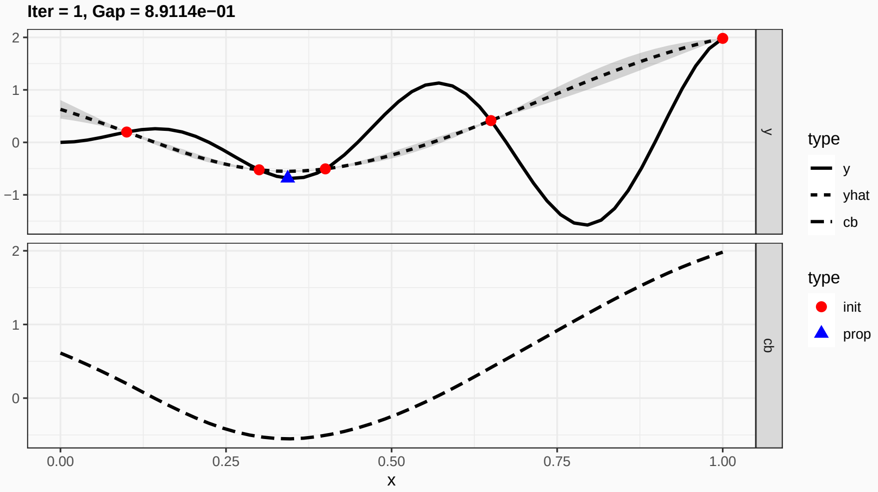
$\tau > 0$ is a constant that controls the “mean vs. uncertainty” trade-off.



The shaded area corresponds to $\hat{\mu}(\mathrm{x}) \pm \tau \cdot s(\mathrm{x})$. Vertical lines are the minimum of $\hat{\mu}(\mathrm{x}) - \tau \cdot s(\mathrm{x})$.

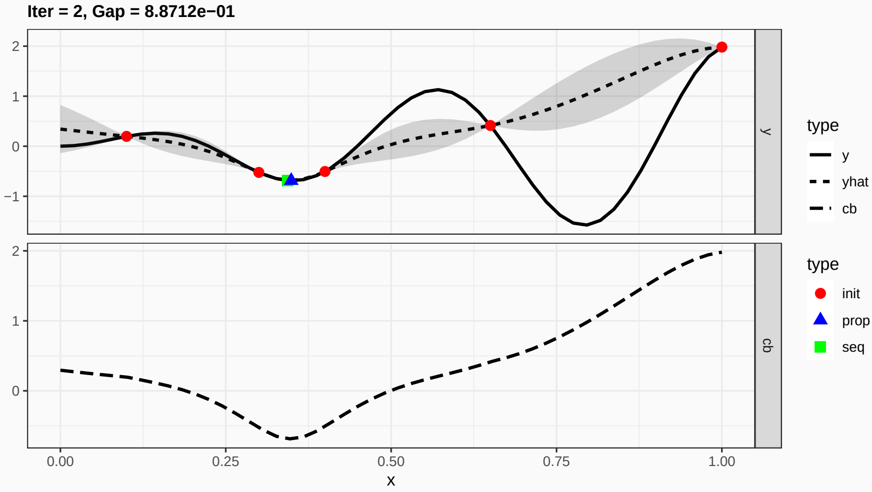
Lower Confidence Bound

The lower τ, the more we focus on pure mean minimization (here $\tau = 0.2$) ...



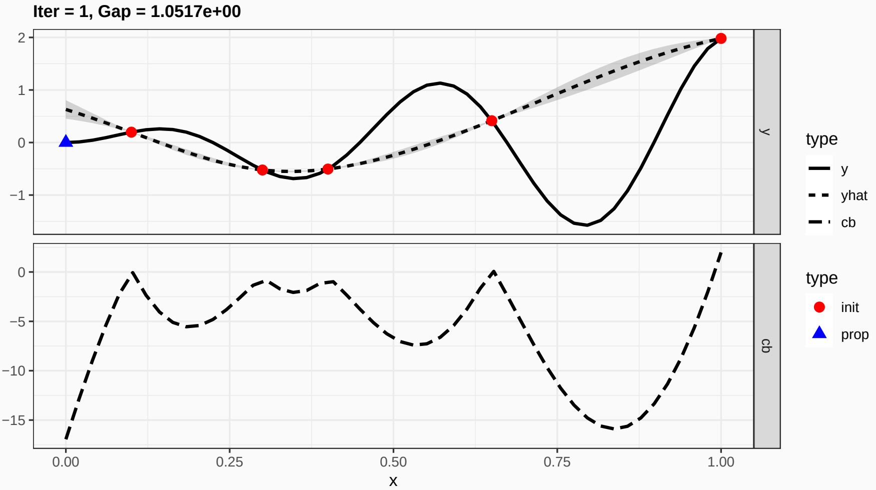
Lower Confidence Bound

The lower τ, the more we focus on pure mean minimization (here $\tau = 0.2$) ...



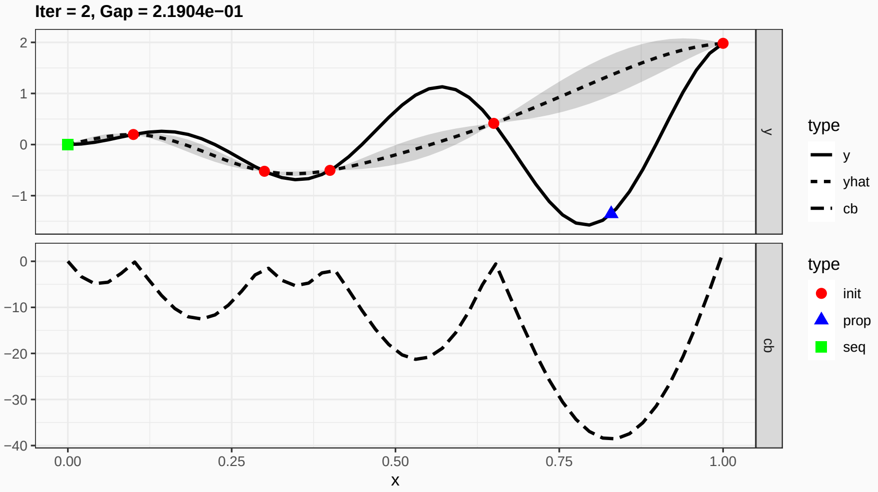
Lower Confidence Bound

... the higher τ, the more we concentrate on reducing variance (here $\tau = 100$).



Lower Confidence Bound

... the higher τ, the more we concentrate on reducing variance (here $\tau = 100$).



Noisy Functions

Noisy Optimization Problem:

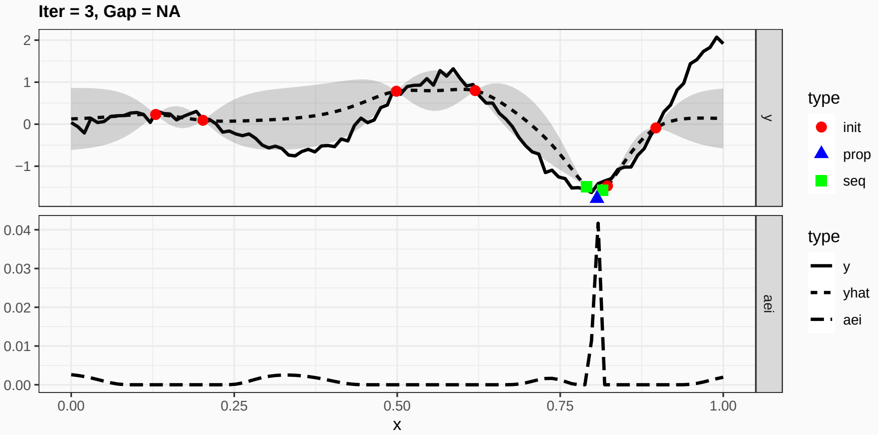
$$y = f(\mathbf{x}) + \epsilon\,, \quad f : \mathcal{X} \to \mathbb{R}$$
$$\epsilon \sim \mathcal{N}(0, \sigma_n^2)$$
$$\mathbf{x}^* := \operatorname{argmin}_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

Consequences:

- EI not valid $\Rightarrow$ use CB or AEI instead.
- Kriging needs to estimate σ_n^2 as well.
- Best observed point will be overoptimistic.

Nugget Estimation

Example with *Kriging* and `nuggest.estim=TRUE` and AEI as acquisition function.



Not Covered

Parallelization

- qCB (exercise!)
- ConstanLiar
- Asynchronous Approaches

Multi-Criteria Optimization

- ParEGO
- SMS-EGO

Common Mistakes

Avoid the following:

- ⚠ Initialize the design wrongly.
    - too small vs. too many factor variables in search space
    - use of wrong y values
- ⚠ Decode (many) factor variables as integer.
- ⚠ No awareness of properties of the surrogate.
    - *Random Forest* does not extrapolate
    - *Random Forest* expresses observational variance
    - *Kriging* depends on the scale / distance
    - *Kriging* expects deterministic outcomes (in default settings)

mlr3tuning Intro

Tuning

- Behavior of most methods depends on *hyperparameters*

Tuning

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well

Tuning

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

Tuning

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

$\Rightarrow$ We do *black box optimization* ("Try stuff and see what works")

Tuning

- Behavior of most methods depends on *hyperparameters*
- We want to choose them so our algorithm performs well
- Good hyperparameters are data-dependent

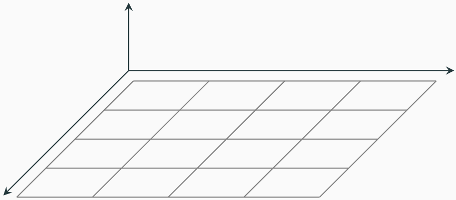
$\Rightarrow$ We do *black box optimization* ("Try stuff and see what works")

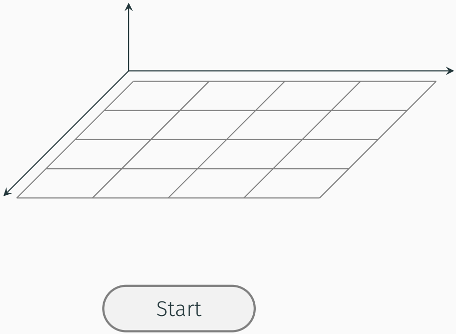
Tuning toolbox for `mlr3`:

```
library("bbotk")
library("mlr3tuning")
```

Tuning

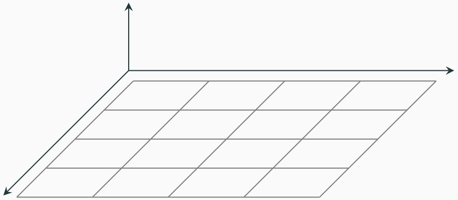
Tuning

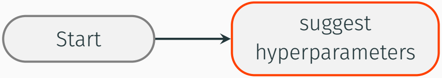




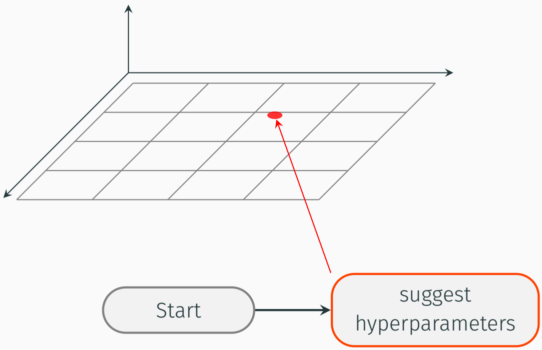
Start

Tuning

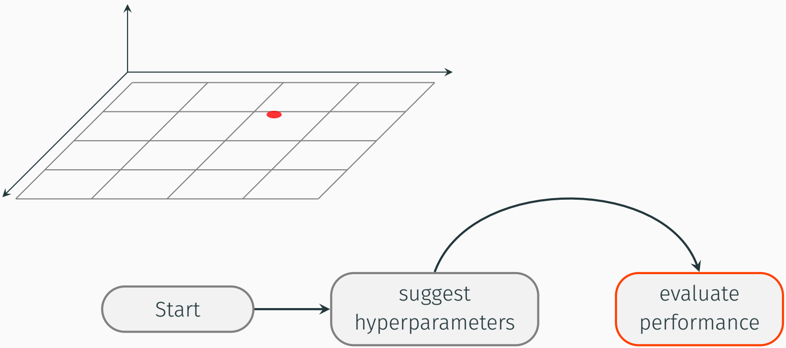




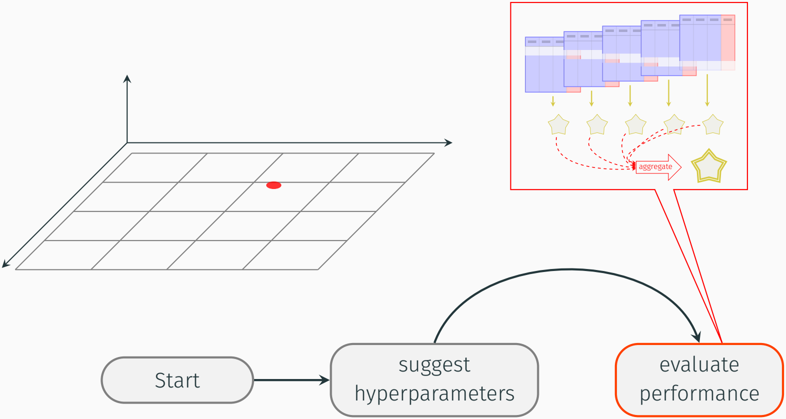
Tuning



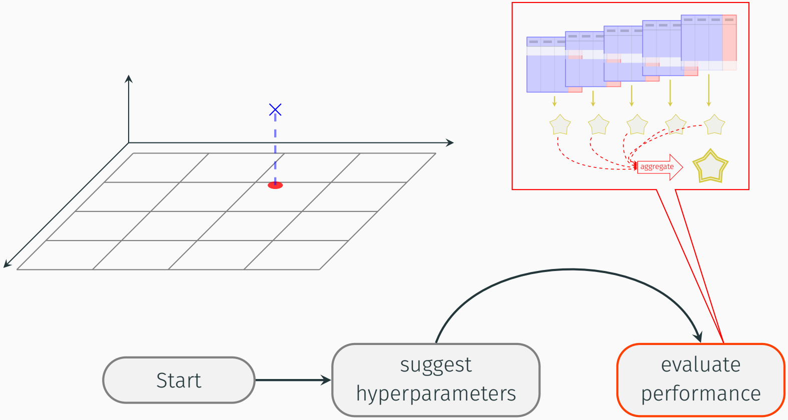
Tuning



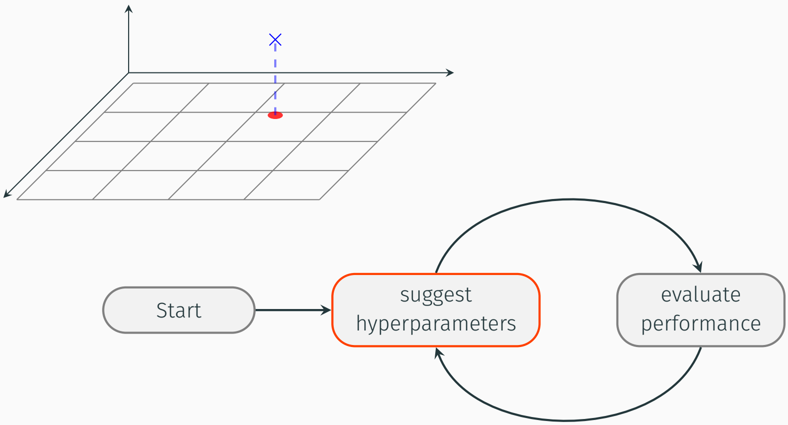
Tuning


aggregate
Start
suggest hyperparameters
evaluate performance

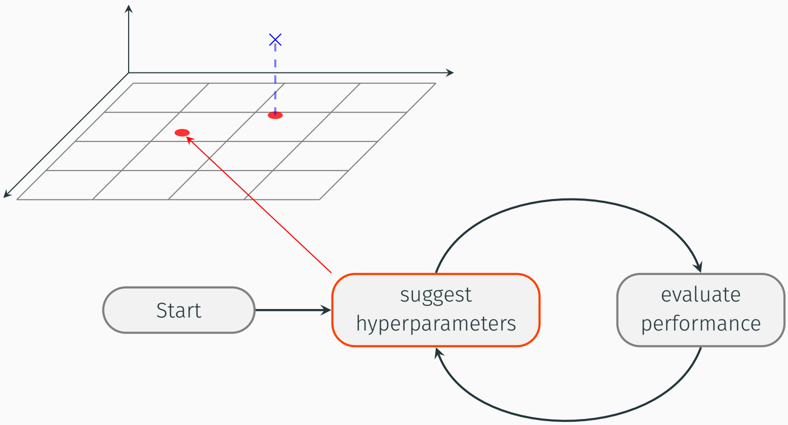
Tuning


aggregate
Start
suggest hyperparameters
evaluate performance

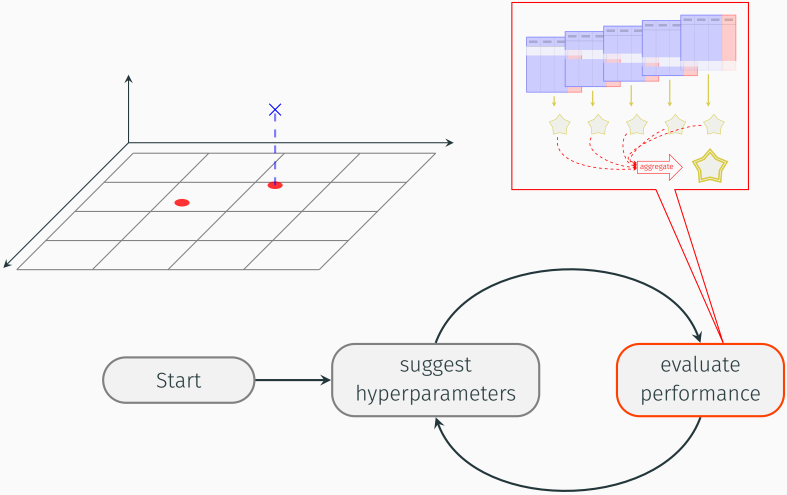
Tuning



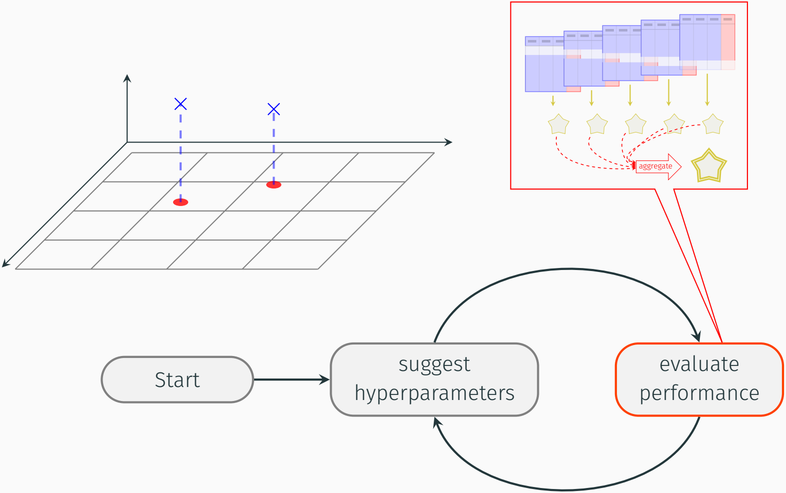
Tuning



Tuning

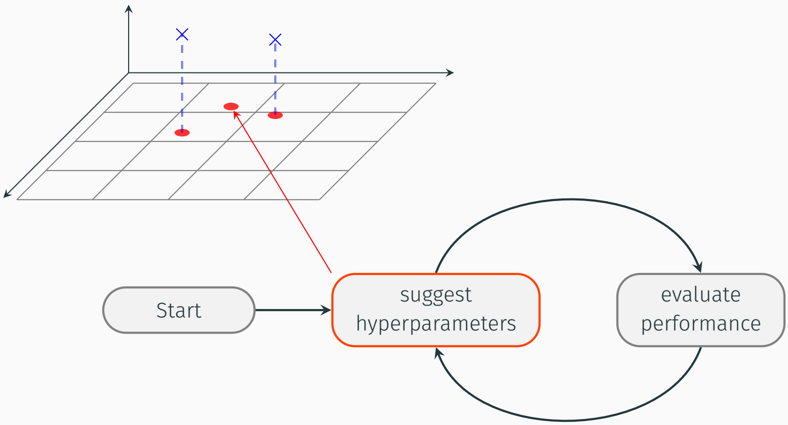

aggregate
Start
suggest hyperparameters
evaluate performance

Tuning

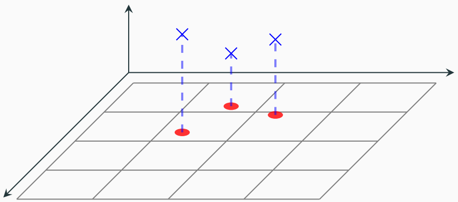


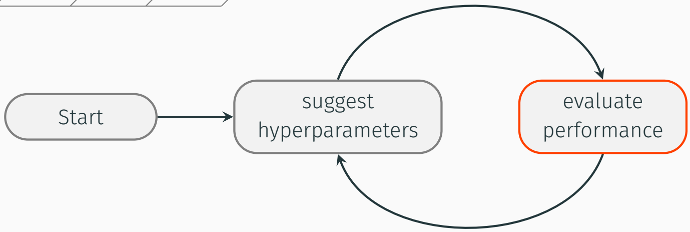
aggregate
Start
suggest hyperparameters
evaluate performance

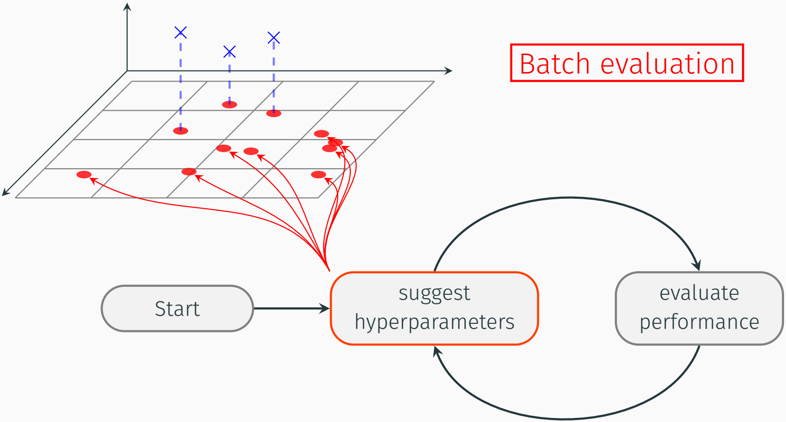
Tuning

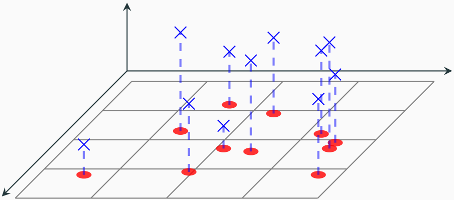


Tuning

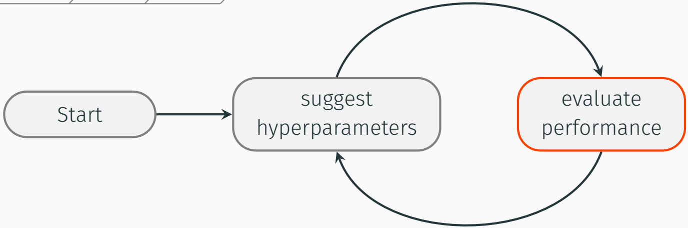




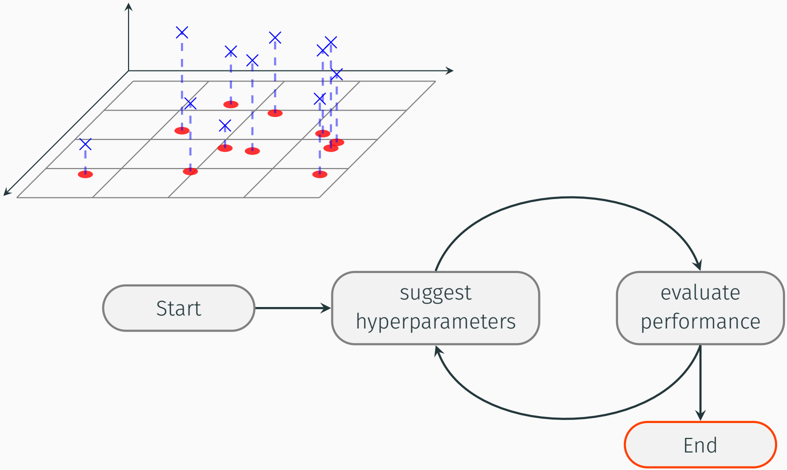

Batch evaluation
Start
suggest hyperparameters
evaluate performance



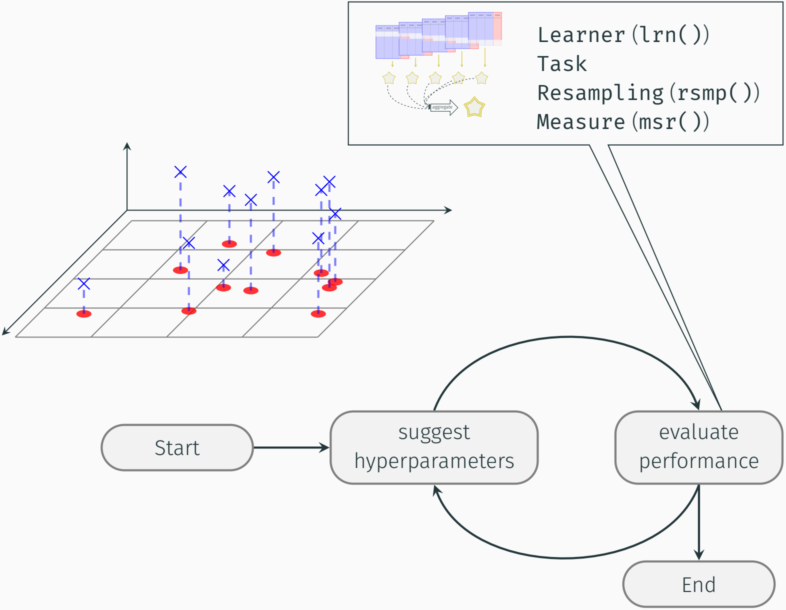
Batch evaluation



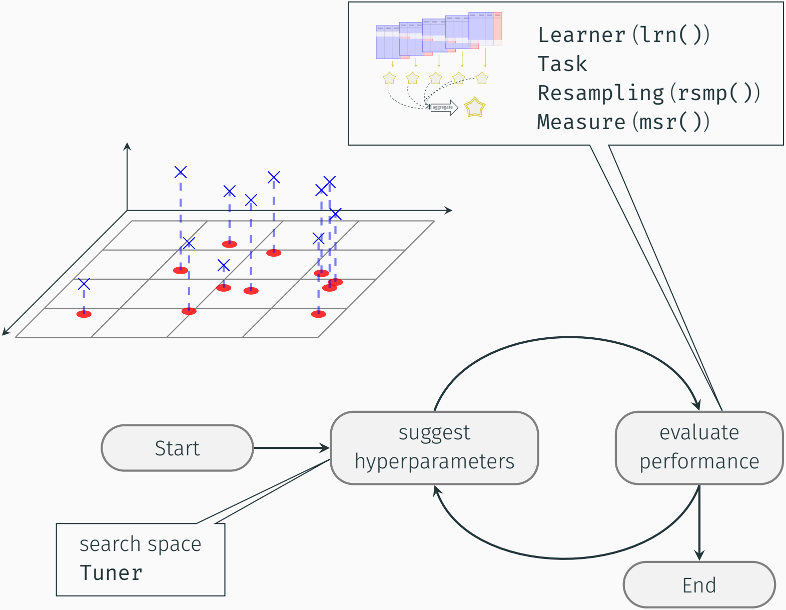
Tuning



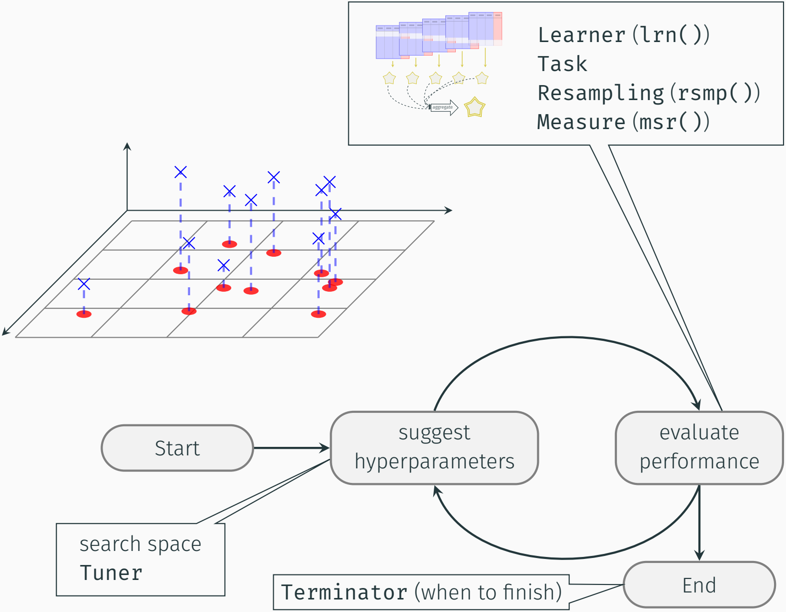
Tuning


Learner (lrn())
Task
Resampling (rsmp())
Measure (msr())
aggregate
Start
suggest hyperparameters
evaluate performance
End

Tuning


Learner (lrn())
Task
Resampling (rsmp())
Measure (msr())
aggregate
Start
suggest hyperparameters
evaluate performance
End
search space
Tuner

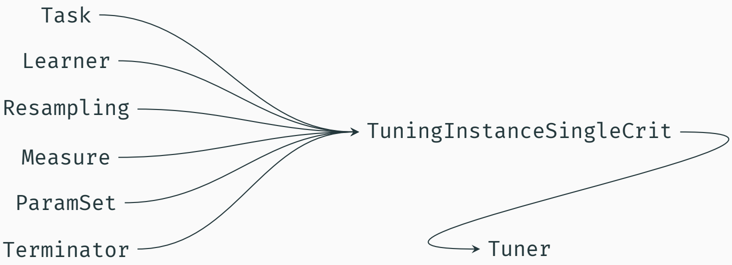
Tuning



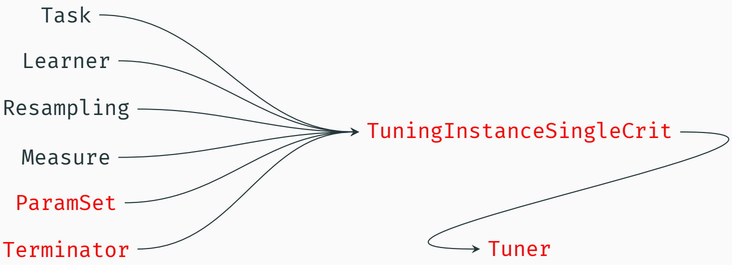
Learner (lrn())
Task
Resampling (rsmp())
Measure (msr())
aggregate
Start
suggest hyperparameters
evaluate performance
End
search space
Tuner
Terminator (when to finish)

Tuning in mlr3

Objects in Tuning



Objects in Tuning



Search Space

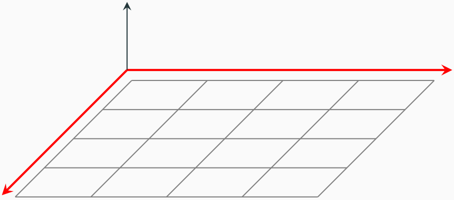
Search Space

```
ParamSet$new(list(param1, param2, ...))
```

Search Space

```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`



```
ParamSet$new(list(param1, param2, ...))
```

Numerical parameter	`ParamDbl$new(id, lower, upper)`
Integer parameter	`ParamInt$new(id, lower, upper)`
Discrete parameter	`ParamFct$new(id, levels)`
Logical parameter	`ParamLgl$new(id)`
Untyped parameter	`ParamUty$new(id)`

```
library("paradox")
searchspace_knn = ParamSet$new(list(
  ParamInt$new("k", 1, 20)
))
```

Termination

- Tuning needs a *termination condition*: when to finish

Termination

- Tuning needs a *termination condition*: when to finish
- `Terminator` class

Termination

- Tuning needs a *termination condition*: when to finish
- `Terminator` class
- `mlr_terminators` dictionary, `trm()` short form

Termination

- Tuning needs a *termination condition*: when to finish
- `Terminator` class
- `mlr_terminators` dictionary, `trm()` short form
-
```
as.data.table(mlr_terminators)
```
```
##                   key
##                &lt;char&gt;
## 1:       clock_time
## 2:            combo
## 3:            evals
## 4:             none
## 5:      perf_reached
## 6:         run_time
## 7:       stagnation
## 8: stagnation_batch
```

Termination

- Tuning needs a *termination condition*: when to finish
- `Terminator` class
- `mlr_terminators` dictionary, `trm()` short form
- ```
  as.data.table(mlr_terminators)
  ```
  ```
  ##                  key
  ##               <char>
  ## 1:       clock_time
  ## 2:            combo
  ## 3:            evals
  ## 4:             none
  ## 5:     perf_reached
  ## 6:         run_time
  ## 7:       stagnation
  ## 8: stagnation_batch
  ```
- ```
  trm("evals", n_evals = 20)
  ```
  ```
  ## <TerminatorEvals>
  ## * Parameters: n_evals=20
  ```

Tuning Method

- need to choose a *tuning method*

Tuning Method

- need to choose a *tuning method*
- `Tuner` class

Tuning Method

- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form

Tuning Method

- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form
-

```
as.data.table(mlr_tuners)
```

```
##               key
##            <char>
## 1: design_points
## 2:         gensa
## 3:   grid_search
## 4:        nloptr
## 5: random_search
```

Tuning Method

- need to choose a *tuning method*
- `Tuner` class
- `mlr_tuners` dictionary, `tnr()` short form
- ```
  as.data.table(mlr_tuners)
  ```
  ```
  ##               key
  ##            <char>
  ## 1: design_points
  ## 2:         gensa
  ## 3:   grid_search
  ## 4:        nloptr
  ## 5: random_search
  ```
- Packages such as `mlr3mbo` extend the available tuners
  ```

Tuning Method

- load `Tuner` with `tnr()`, set parameters

Tuning Method

- load `Tuner` with `tnr()`, set parameters
- ```
  gsearch = tnr("grid_search", resolution = 3)

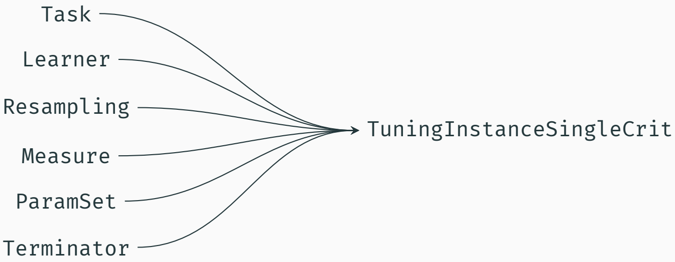
  print(gsearch)
  ## <TunerGridSearch>
  ## * Parameters: resolution=3, batch_size=1
  ## * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
  ## * Properties: dependencies, single-crit, multi-crit
  ## * Packages: -
  ```
  ```

Tuning Method

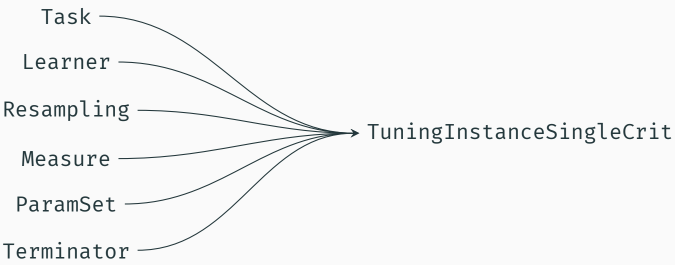
- load `Tuner` with `tnr()`, set parameters
- ```
  gsearch = tnr("grid_search", resolution = 3)

  print(gsearch)
  ## <TunerGridSearch>
  ## * Parameters: resolution=3, batch_size=1
  ## * Parameter classes: ParamLgl, ParamInt, ParamDbl, ParamFct
  ## * Properties: dependencies, single-crit, multi-crit
  ## * Packages: -
  ```
- common parameter `batch_size` for parallelization
  ```

Calling the Tuner



Calling the Tuner




```r
inst = TuningInstanceSingleCrit$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("holdout"), msr("classif.ce"),
  searchspace_knn, trm("none")
)
```

Calling the Tuner

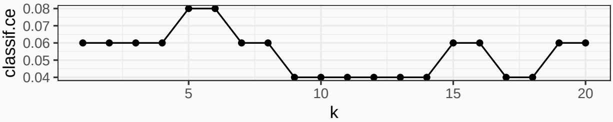
```
gsearch$optimize(inst)

##          k learner_param_vals  x_domain classif.ce
##      <num>             <list>    <list>      <num>
## 1:      10          <list[2]> <list[1]>       0.04
```

Tuning Results

```r
gsearch = tnr("grid_search", resolution = 20)

inst = TuningInstanceSingleCrit$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"), rsmp("holdout"),
  msr("classif.ce"),searchspace_knn, trm("none"))

gsearch$optimize(inst)
```

```
##        k learner_param_vals  x_domain classif.ce
##    <num>             <list>    <list>      <num>
## 1:    11          <list[2]> <list[1]>       0.04
```

```r
ggplot(inst$archive$data(),
  aes(x = k, y = classif.ce)) + geom_line() + geom_point()
```



Recap

```
inst = TuningInstanceSingleCrit$new(
  tsk("iris"), lrn("classif.kknn", kernel="rectangular"),
  rsmp("holdout"), msr("classif.ce"),
  searchspace_knn, trm("evals", n_evals = 2)
)

gsearch = tnr("grid_search", resolution = 3)

gsearch$optimize(inst)

##        k learner_param_vals  x_domain classif.ce
##    <num>             <list>    <list>      <num>
## 1:     1          <list[2]> <list[1]>       0.04
```

Parameter Transformation

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. optimize from $\log(1) \ldots \log(100)$ (`k_before_trafo`)

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. optimize from $\log(1) \ldots \log(100)$ (`k_before_trafo`)
2. transform by $\exp()$ in `trafo` function

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

- Part of `ParamSet`

Example:

1. optimize from $\log(1) \dots \log(100)$ (`k_before_trafo`)
2. transform by $\exp()$ in `trafo` function
3. don't forget to `round` (k must be integer)

Parameter Transformation

- Sometimes we do not want to optimize over an evenly spaced range
- $k = 1$ vs. $k = 2$ probably more interesting than $k = 101$ vs. $k = 102$

$\Rightarrow$ Transformations

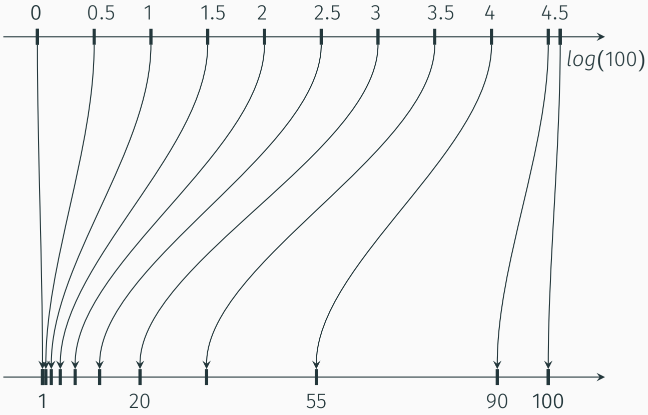
- Part of `ParamSet`

Example:

1. optimize from $\log(1) \ldots \log(100)$ (`k_before_trafo`)
2. transform by $\exp()$ in `trafo` function
3. don't forget to `round` (k must be integer)

```
searchspace_knn_trafo = ParamSet$new(list(
  ParamDbl$new("k_before_trafo", log(1), log(50))
))
searchspace_knn_trafo$trafo = function(x, param_set) {
  return(list(k = round(exp(x$k_before_trafo))))
}
```

What is our transformation doing?

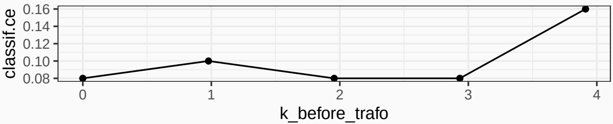


Tuning again...

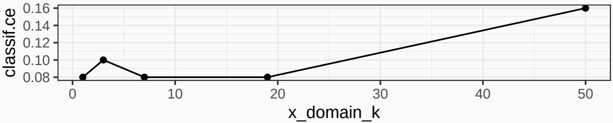
```
inst$result

##    k_before_trafo learner_param_vals  x_domain classif.ce
##             <num>             <list>    <list>      <num>
## 1:            2.9          <list[2]> <list[1]>       0.08

inst$result$x_domain

## [[1]]
## [[1]]$k
## [1] 19
```

Parameter Transformation

```r
ggplot(inst$archive$data(),
  aes(x = k_before_trafo, y = classif.ce)) + geom_line() + geom_point()
```




```r
ggplot(inst$archive$data(unnest = "x_domain"),
  aes(x = x_domain_k, y = classif.ce)) + geom_line() + geom_point()
```

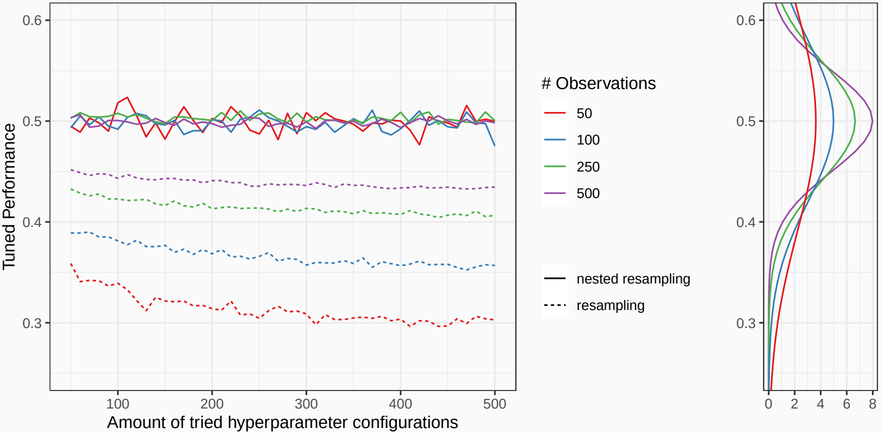


Nested Resampling

Nested Resampling - Instructive Example

Tuning a hyperparameter that does not have any effect also shows imaginary "tuning sucess".

```
## `summarise()` regrouping output by 'data_dim', 'tuning_method'
(override with `.groups` argument)
```

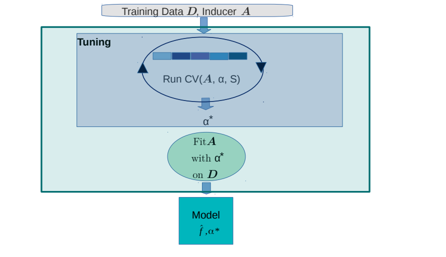


Nested Resampling

- Need to perform nested resampling to estimate tuned learner performance

$\Rightarrow$ Treat tuning as if it were a `Learner`!

- Training:
  1. Tune model using (inner) resampling
  2. Train final model with best parameters on all (i.e. outer resampling) data
- Predicting: Just use final model



Nested Resampling

```r
optlrn = AutoTuner$new(lrn("classif.kknn", kernel="rectangular"),
  rsmp("holdout"), msr("classif.ce"), searchspace_knn,
  trm("none"), tnr("grid_search", resolution = 10))
```

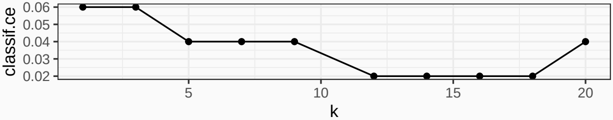
```r
optlrn$train(tsk("iris"))
```

```r
optlrn$model$learner

## Learner classif.kknn from package
## Type:
## Name: ; Short name:
## Class: LearnerClassifKKNN
## Properties: multiclass,twoclass
## Predict-Type:
## Hyperparameters:
```

Nested Resampling

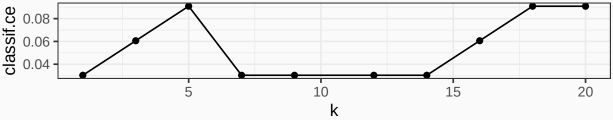
Performance observed during tuning on the complete dataset.

```
ggplot(optlrn$model$tuning_instance$archive$data(),
 aes(x = k, y = classif.ce)) + geom_line() + geom_point()
```



Performance observed during tuning on the tuning dataset.

```r
result = resample(tsk("iris"), optlrn, rsmp("holdout"),
  store_models = TRUE)
ggplot(result$learners[[1]]$
  model$tuning_instance$archive$data(),
  aes(x = k, y = classif.ce)) + geom_line() + geom_point()
```



mlr3tuning recap

Tuning with mlr3tuning

Tuning a `Learner`

1. Construct a `TuningInstanceSingleCrit`
   - `Task`—the Data to tune over
   - `Learner`—the algorithm to tune
   - `Resampling`—the resampling method to use
   - `Measure`—how to evaluate performance
   - `ParamSet`—the search space, possibly with `trafo`
   - `Terminator`—when to quit
2. Create a `Tuner`
   - Usually using `tnr()`
   - May have some parameters, e.g. `batch_size`
3. Call `tuner$optimize()`

Nested Resampling

1. Construct an `AutoTuner`
   - Constructor takes all arguments of a `TuningInstanceSingleCrit` *except* `Task`
   - Also takes the `Tuner` as an argument
2. Use like a normal `Learner` in `resample()` and `benchmark()`

mlr3mbo Conclusion

mlr3mbo Conclusion

Key features

- Highly customizable expensive Black-Box optimization
- Integrated parallelization
- Multi-objective optimization
- Seamless `mlr3` integration

Resources

Help:

- `https://mlr-org.github.io/mlr3mbo` ⇒ under construction
- `https://mlr3book.mlr-org.com/`

Bug + Issue Tracker: `https://github.com/mlr-org/mlr3mbo/issues`

Mattermost Chanel #mlr3mbo:
`https://lmmisld-lmu-stats-slds.srv.mwn.de/mlr/mlr3mbo`