



HYPERPARAMETER TUNING IN R

# Machine learning with mlr

Dr. Shirin Glander  
Data Scientist



# The mlr package

- `mlr` is another framework for **machine learning** in R.

Model training follows **three steps**:

1. Define the **task**
2. Define the **learner**
3. Fit the **model**

<https://mlr-org.github.io/mlr>

# New dataset: User Knowledge Data

```
library(tidyverse)
glimpse(knowledge_data)

Observations: 150
Variables: 6
$ STG <dbl> 0.080, 0.000, 0.180, 0.100, 0.120, 0.090, 0.080, 0.150, ...
$ SCG <dbl> 0.080, 0.000, 0.180, 0.100, 0.120, 0.300, 0.325, 0.275, ...
$ STR <dbl> 0.100, 0.500, 0.550, 0.700, 0.750, 0.680, 0.620, 0.800, ...
$ LPR <dbl> 0.24, 0.20, 0.30, 0.15, 0.35, 0.18, 0.94, 0.21, 0.19, ...
$ PEG <dbl> 0.90, 0.85, 0.81, 0.90, 0.80, 0.85, 0.56, 0.81, 0.82, ...
$ UNS <chr> "High", "High", "High", "High", "High", "High", "High", ...

knowledge_data %>%
  count(UNS)

# A tibble: 3 x 2
  UNS      n
  <chr> <int>
1 High    50
2 Low     50
3 Middle  50
```

# Tasks in mlr for supervised learning

- `RegrTask()` for regression
- `ClassifTask()` for binary and multi-class classification
- `MultilabelTask()` for multi-label classification problems
- `CostSensTask()` for general cost-sensitive classification

With our student knowledge dataset we can build a classifier:

```
task <- makeClassifTask(data = knowledge_train_data,  
                        target = "UNS")
```



# Learners in mlr

```
listLearners()

      class      package
1      classif.ada      ada, rpart
2      classif.adaboostml      RWeka
3      classif.bartMachine      bartMachine
4      classif.binomial      stats
5      classif.boosting      adabag, rpart
6      classif.bst      bst, rpart
7      classif.C50      C50
8      classif.cforest      party
9      classif.clusterSVM      SwarmSVM, Liblinear, R
10     classif.ctree      party
...

# Define learner
lrn <- makeLearner("classif.h2o.deeplearning",
                  fix.factors.prediction = TRUE,
                  predict.type = "prob")
```



# Model fitting in mlr

```
tic()
# Define task
task <- makeClassifTask(data = knowledge_train_data,
                        target = "UNS")

# Define learner
lrn <- makeLearner("classif.h2o.deeplearning",
                  fix.factors.prediction = TRUE)

# Fit model
model <- train(lrn,
               task)

toc()

3.782 sec elapsed
```



## HYPERPARAMETER TUNING IN R

**Let's practice!**



HYPERPARAMETER TUNING IN R

# Hyperparameter tuning with mlr - grid and random search

Dr. Shirin Glander  
Data Scientist





# Hyperparameter tuning with mlr

In `mlr` you have to define

1. the **search space** for every hyperparameter
2. the **tuning method** (e.g. grid or random search)
3. the **resampling method**

# Defining the search space

```
makeParamSet(  
  makeNumericParam(),  
  makeIntegerParam(),  
  makeDiscreteParam(),  
  makeLogicalParam(),  
  makeDiscreteVectorParam()  
)
```

```
getParamSet("classif.h2o.deeplearning")
```

	Type	len	Def
autoencoder	logical	-	FALSE
use_all_factor_level	logical	-	TRUE
activation	discrete	-	Rectifier
hidden	integervector	<NA>	200,200
epochs	numeric	-	10
train_samples_per_iteration	numeric	-	-2
seed	integer	-	-
adaptive_rate	logical	-	TRUE
rho	numeric	-	0.99
epsilon	numeric	-	1e-08
rate	numeric	-	0.005

# Defining the search space

```
getParamSet("classif.h2o.deeplearning")
```

	Type	len	Def
autoencoder	logical	-	FALSE
use_all_factor_level	logical	-	TRUE
activation	discrete	-	Rectifier
hidden	integervector	<NA>	200,200
epochs	numeric	-	10
train_samples_per_iteration	numeric	-	-2
seed	integer	-	-
adaptive_rate	logical	-	TRUE
rho	numeric	-	0.99
epsilon	numeric	-	1e-08
rate	numeric	-	0.005

```
param_set <- makeParamSet(  
  makeDiscreteParam("hidden", values = list(one = 10, two = c(10, 5, 10))),  
  makeDiscreteParam("activation", values = c("Rectifier", "Tanh")),  
  makeNumericParam("l1", lower = 0.0001, upper = 1),  
  makeNumericParam("l2", lower = 0.0001, upper = 1)  
)
```

# Defining the tuning method

- Grid search

```
ctrl_grid <- makeTuneControlGrid()  
ctrl_grid
```

```
Tune control: TuneControlGrid  
Same resampling instance: TRUE  
Imputation value: <worst>  
Start: <NULL>
```

```
Tune threshold: FALSE  
Further arguments: resolution=10
```

- Random search

```
ctrl_random <- makeTuneControlRandom()  
ctrl_random
```

```
Tune control: TuneControlRandom  
Same resampling instance: TRUE  
Imputation value: <worst>  
Start: <NULL>
```

```
Budget: 100  
Tune threshold: FALSE  
Further arguments: maxit=100
```

Can only deal with **discrete** parameter sets!

# Define resampling strategy

```
cross_val <- makeResampleDesc("RepCV",  
                             predict = "both",  
                             folds = 5 * 3)  
  
param_set <- makeParamSet(  
  ...  
)  
  
ctrl_grid <- makeTuneControlGrid()  
  
task <- makeClassifTask(data = knowledge_train_data,  
                        target = "UNS")  
  
lrn <- makeLearner("classif.h2o.deeplearning",  
                  predict.type = "prob",  
                  fix.factors.prediction = TRUE)  
  
lrn_tune <- tuneParams(lrn,  
                       task,  
                       resampling = cross_val,  
                       control = ctrl_grid,  
                       par.set = param_set)
```

# Tuning hyperparameters

```
lrn_tune <- tuneParams(lrn,
                      task,
                      resampling = cross_val,
                      control = ctrl_grid,
                      par.set = param_set)

[Tune-y] 27: mmce.test.mean=0.6200000; time: 0.0 min
[Tune-x] 28: hidden=two; activation=Rectifier; l1=0.578; l2=1
[Tune-y] 28: mmce.test.mean=0.6800000; time: 0.0 min
[Tune-x] 29: hidden=one; activation=Rectifier; l1=0.156; l2=0.68
[Tune-y] 29: mmce.test.mean=0.4400000; time: 0.0 min
[Tune-x] 30: hidden=one; activation=Rectifier; l1=0.717; l2=0.427
[Tune-y] 30: mmce.test.mean=0.6600000; time: 0.0 min

[Tune] Result: hidden=two; activation=Tanh; l1=0.113;
l2=0.0973 : mmce.test.mean=0.2000000

# tictoc
26.13 sec elapsed
```



## HYPERPARAMETER TUNING IN R

**Let's practice!**



HYPERPARAMETER TUNING IN R

# Evaluating tuned hyperparameters with mlr

Dr. Shirin Glander  
Data Scientist





# Evaluation of our results can tell us:

- How different hyperparameters **affect the performance** of our model.
- Which hyperparameters have a particularly strong or weak **impact** on our model performance.
- Whether our hyperparameter search **converged**, i.e. whether we can be reasonably confident that we found the most **optimal hyperparameter combination** (or close to it).

# Recap

```
getParamSet("classif.h2o.deeplearning")

param_set <- makeParamSet(
  makeDiscreteParam("hidden", values = list(one = 10, two = c(10, 5, 10))),
  makeDiscreteParam("activation", values = c("Rectifier", "Tanh")),
  makeNumericParam("l1", lower = 0.0001, upper = 1),
  makeNumericParam("l2", lower = 0.0001, upper = 1)
)

ctrl_random <- makeTuneControlRandom(maxit = 50)

holdout <- makeResampleDesc("Holdout")

task <- makeClassifTask(data = knowledge_train_data, target = "UNS")

lrn <- makeLearner("classif.h2o.deeplearning", predict.type = "prob",
  fix.factors.prediction = TRUE)

lrn_tune <- tuneParams(lrn,
  task,
  resampling = holdout,
  control = ctrl_random,
  par.set = param_set)
```

# Evaluating the tuning results

```
lrn_tune
```

```
Tune result:
```

```
Op. pars: hidden=one; activation=Rectifier; l1=0.541; l2=0.229
```

```
mmce.test.mean=0.160000
```

```
generateHyperParsEffectData(lrn_tune, partial.dep = TRUE)
```

```
HyperParsEffectData:
```

```
Hyperparameters: hidden,activation,l1,l2
```

```
Measures: mmce.test.mean
```

```
Optimizer: TuneControlRandom
```

```
Nested CV Used: FALSE
```

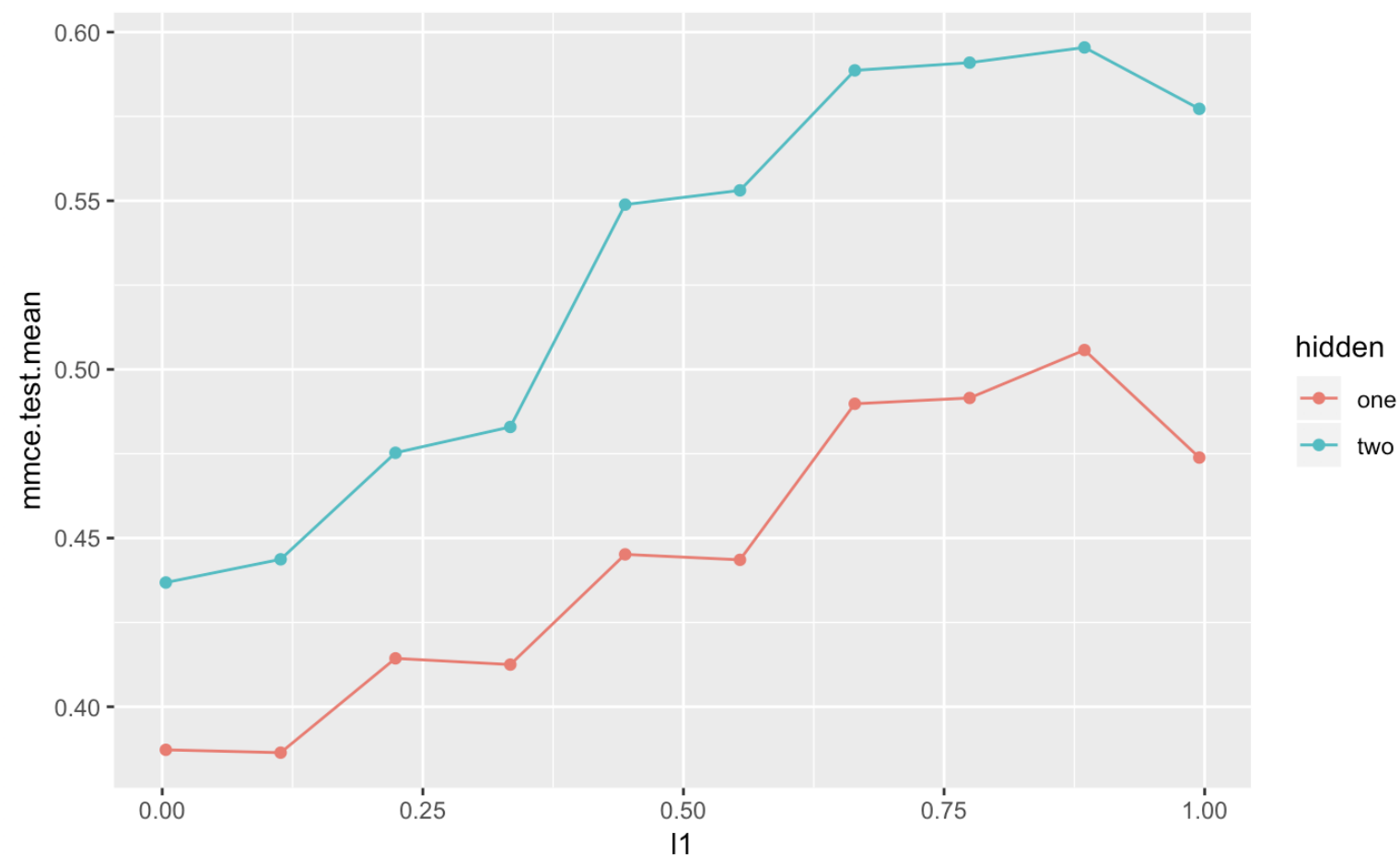
```
[1] "Partial dependence requested"
```

```
Snapshot of data:
```

	hidden	activation	l1	l2	mmce.test.mean	iteration	exec.time
1	one	Rectifier	0.75940339	0.9956819	0.40	1	0.883
2	one	Rectifier	0.16701526	0.2948697	0.40	2	0.836
3	one	Rectifier	0.88458832	0.9228281	0.70	3	0.830
4	two	Rectifier	0.48840740	0.7276899	0.70	4	0.820
5	one	Tanh	0.87114452	0.9971268	0.40	5	0.835
6	two	Tanh	0.07412213	0.3841913	0.44	6	0.830

# Plotting hyperparameter tuning results

```
hyperpar_effects <- generateHyperParsEffectData(lrn_tune, partial.dep = TRUE)
plotHyperParsEffect(hyperpar_effects, partial.dep.learn = "regr.randomForest",
  x = "l1", y = "mmce.test.mean", z = "hidden",
  plot.type = "line")
```





## HYPERPARAMETER TUNING IN R

**Now it's your turn!**



HYPERPARAMETER TUNING IN R

# Advanced tuning with mlr

**Dr. Shirin Glander**  
Data Scientist



# Advanced tuning controls

- `makeTuneControlCMAES`: **CMA Evolution Strategy**
- `makeTuneControlDesign`: **Predefined data frame of hyperparameters**
- `makeTuneControlGenSA`: **Generalized simulated annealing**
- `makeTuneControlIrace`: **Tuning with iterated F-Racing**
- `makeTuneControlMBO`: **Model-based / Bayesian optimization**

# Choosing evaluation metrics

```
# Generalized simulated annealing
ctrl_gensa <- makeTuneControlGenSA()

# Create holdout sampling
bootstrap <- makeResampleDesc("Bootstrap", predict = "both")

# Perform tuning
lrn_tune <- tuneParams(learner = lrn,
                      task = task,
                      resampling = bootstrap,
                      control = ctrl_gensa,
                      par.set = param_set,
                      measures = list(acc, mmce))

[Tune-x] 2170: eta=0.0771; max_depth=4
[Tune-y] 2170: acc.test.mean=0.9317275,mmce.test.mean=0.0682725; time: 0.0 m
[Tune-x] 2171: eta=0.822; max_depth=8
[Tune-y] 2171: acc.test.mean=0.9276912,mmce.test.mean=0.0723088; time: 0.0 m
[Tune-x] 2172: eta=0.498; max_depth=4
[Tune-y] 2172: acc.test.mean=0.9311626,mmce.test.mean=0.0688374; time: 0.0 m
[Tune-x] 2173: eta=0.365; max_depth=4
[Tune-y] 2173: acc.test.mean=0.9288406,mmce.test.mean=0.0711594; time: 0.0 m
```



# Choosing evaluation metrics

```
# Create holdout sampling
bootstrap <- makeResampleDesc("Bootstrap", predict = "both")

# Perform tuning
lrn_tune <- tuneParams(learner = lrn,
                      task = task,
                      resampling = bootstrap,
                      control = ctrl_gensa,
                      par.set = param_set,
                      measures = list(acc,
                                     setAggregation(acc, train.mean),
                                     mmce,
                                     setAggregation(mmce, train.mean)))

[Tune-x] 3920: eta=0.294; max_depth=8
[Tune-y] 3920: acc.test.mean=0.9250118,
              acc.train.mean=0.9740000,
              mmce.test.mean=0.0749882,
              mmce.train.mean=0.0260000;
              time: 0.0 min
```

# Nested cross-validation & nested resampling

```
lrn_wrapper <- makeTuneWrapper(learner = lrn,  
                               resampling = bootstrap,  
                               control = ctrl_gensa,  
                               par.set = param_set,  
                               measures = list(acc, mmce))
```

- Either train **directly**

```
model_nested <- train(lrn_wrapper, task)  
getTuneResult(model_nested)
```

- Or add 2x **nested** cross-validation

```
cv2 <- makeResampleDesc("CV", iters = 2)  
res <- resample(lrn_wrapper,  
                task,  
                resampling = cv2,  
                extract = getTuneResult)  
generateHyperParsEffectData(res)
```

# Choose hyperparameters from a tuning set

```
lrn_best <- setHyperPars(lrn, par.vals = list(minsplit = 4,  
                                             minbucket = 3,  
                                             maxdepth = 6))
```

```
model_best <- train(lrn_best, task)
```

```
predict(model_best, newdata = knowledge_test_data)
```

```
Prediction: 30 observations
```

```
predict.type: response
```

```
threshold:
```

```
time: 0.00
```

```
  truth response
```

```
1  High      High
```

```
2  High      High
```

```
3  High      High
```

```
4  High      High
```

```
5  High      High
```

```
6  High      High
```

```
... (#rows: 30, #cols: 2)
```



## HYPERPARAMETER TUNING IN R

**It's your turn!**