Machine Learning with R at LRZ: Introduction to mlr

Resampling

We will continue our example for spam classification

a) Instead of manually splitting train and test set create a holdout set directly in mlr. Use the set to evaluate the performance of an algorithm of your choice on the spam data. Use 80% of the data for training and create stratified splits.

```
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```
data(spam, package = "kernlab")
spam.task = makeClassifTask(data = spam, target = "type")
lrn = makeLearner("classif.rpart", predict.type = "prob")
```

b) Now create a 10-fold crossvalidation and evaluate AUC and training time

```
resample(lrn, spam.task, cv10, measures = list(auc, timetrain))
```

```
## Resampling: cross-validation
## Measures:             auc         timetrain
## [Resample] iter 1:    0.8829      0.0640
## [Resample] iter 2:    0.8859      0.0480
## [Resample] iter 3:    0.8877      0.0520
## [Resample] iter 4:    0.8886      0.0500
## [Resample] iter 5:    0.8961      0.0540
## [Resample] iter 6:    0.8898      0.1050
## [Resample] iter 7:    0.8645      0.0510
## [Resample] iter 8:    0.8824      0.0490
## [Resample] iter 9:    0.9117      0.0430
## [Resample] iter 10:   0.9104      0.0530
##
## Aggregated Result: auc.test.mean=0.8900,timetrain.test.mean=0.0569
##
## Resample Result
## Task: spam
## Learner: classif.rpart
## Aggr perf: auc.test.mean=0.8900,timetrain.test.mean=0.0569
## Runtime: 0.67017
```

Benchmarking

We would like to create a small benchmark study to see how much complexity is required to achieve an AUC of at least 98%.

a) Create the following learning algorithms to compare their performance

- Featureless baseline learner
- Linear Discriminant Analysis
- Logistic Regression
- Classification Tree
- Random Forest

```
lrns = makeLearners(type = "classif", c("featureless", "lda", "logreg", "rpart",
                                        "randomForest"), predict.type = "prob")
lrns
```

```
## $classif.featureless
## Learner classif.featureless from package mlr
## Type: classif
## Name: Featureless classifier; Short name: featureless
## Class: classif.featureless
## Properties: twoclass,multiclass,numerics,factors,ordered,missings,prob,functionals
## Predict-Type: prob
## Hyperparameters:
##
##
## $classif.lda
## Learner classif.lda from package MASS
## Type: classif
## Name: Linear Discriminant Analysis; Short name: lda
## Class: classif.lda
## Properties: twoclass,multiclass,numerics,factors,prob
## Predict-Type: prob
## Hyperparameters:
##
##
## $classif.logreg
## Learner classif.logreg from package stats
## Type: classif
## Name: Logistic Regression; Short name: logreg
## Class: classif.logreg
## Properties: twoclass,numerics,factors,prob,weights
## Predict-Type: prob
## Hyperparameters: model=FALSE
##
##
## $classif.rpart
## Learner classif.rpart from package rpart
## Type: classif
## Name: Decision Tree; Short name: rpart
## Class: classif.rpart
## Properties: twoclass,multiclass,missings,numerics,factors,ordered,prob,weights,featimp
## Predict-Type: prob
## Hyperparameters: xval=0
##
##
## $classif.randomForest
## Learner classif.randomForest from package randomForest
## Type: classif
```

```
## Name: Random Forest; Short name: rf
## Class: classif.randomForest
## Properties: twoclass,multiclass,numerics,factors,ordered,prob,class.weights,oobpreds,featimp
## Predict-Type: prob
## Hyperparameters:
```

b) Benchmark the five learning algorithms with a 5-fold crossvalidation (ensure identical folds for all learners). Measure the AUC as well as the runtime.

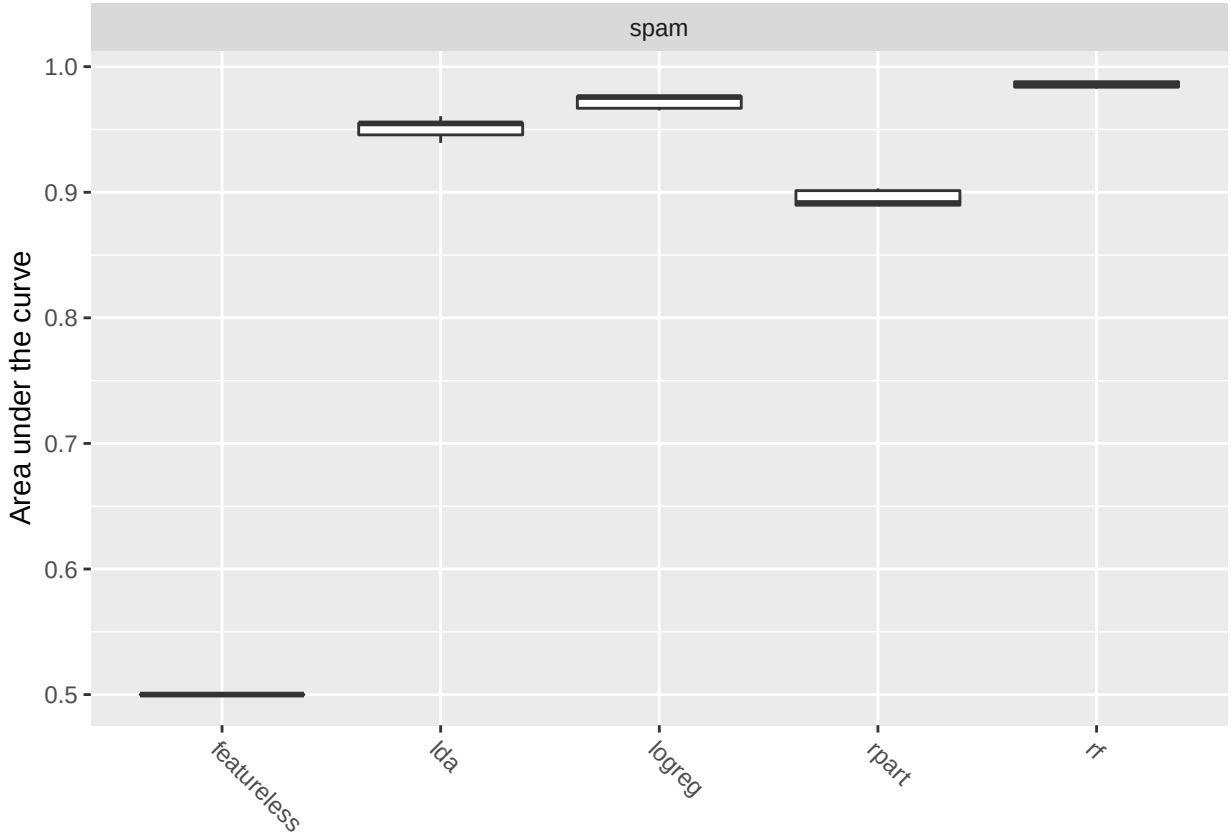
```
bmr = benchmark(lrns, spam.task, cv5, measures = list(auc, timetrain))
```

```
## Task: spam, Learner: classif.featureless
## Resampling: cross-validation
## Measures:             auc         timetrain
## [Resample] iter 1:    0.5000      0.0000
## [Resample] iter 2:    0.5000      0.0000
## [Resample] iter 3:    0.5000      0.0000
## [Resample] iter 4:    0.5000      0.0000
## [Resample] iter 5:    0.5000      0.0000
##
## Aggregated Result: auc.test.mean=0.5000,timetrain.test.mean=0.0000
##
## Task: spam, Learner: classif.lda
## Resampling: cross-validation
## Measures:             auc         timetrain
## [Resample] iter 1:    0.9551      0.2120
## [Resample] iter 2:    0.9393      0.0690
## [Resample] iter 3:    0.9548      0.0730
## [Resample] iter 4:    0.9457      0.0740
## [Resample] iter 5:    0.9606      0.0700
##
## Aggregated Result: auc.test.mean=0.9511,timetrain.test.mean=0.0996
##
## Task: spam, Learner: classif.logreg
## Resampling: cross-validation
## Measures:             auc         timetrain
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## [Resample] iter 1:    0.9763      0.3410
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## [Resample] iter 2:    0.9669      0.3290
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

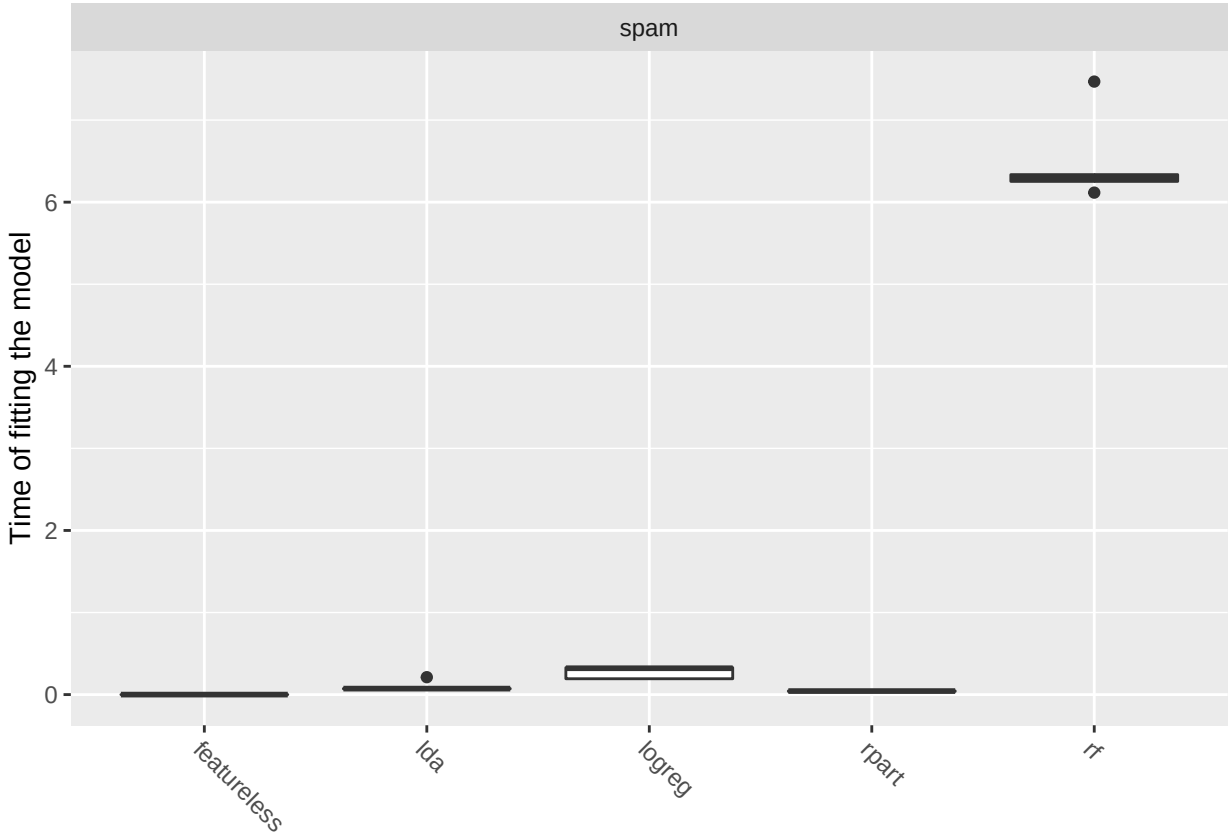
```
## [Resample] iter 3:    0.9758      0.1910
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## [Resample] iter 4:    0.9652      0.1930
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## [Resample] iter 5:    0.9768      0.3210
##
## Aggregated Result: auc.test.mean=0.9722,timetrain.test.mean=0.2750
##
## Task: spam, Learner: classif.rpart
## Resampling: cross-validation
## Measures:             auc         timetrain
## [Resample] iter 1:    0.8897      0.0390
## [Resample] iter 2:    0.8915      0.0450
## [Resample] iter 3:    0.9013      0.0430
## [Resample] iter 4:    0.9030      0.0450
## [Resample] iter 5:    0.8897      0.0380
##
## Aggregated Result: auc.test.mean=0.8950,timetrain.test.mean=0.0420
##
## Task: spam, Learner: classif.randomForest
## Resampling: cross-validation
## Measures:             auc         timetrain
## [Resample] iter 1:    0.9851      6.3350
## [Resample] iter 2:    0.9825      7.4690
## [Resample] iter 3:    0.9838      6.2900
## [Resample] iter 4:    0.9880      6.2530
## [Resample] iter 5:    0.9881      6.1160
##
## Aggregated Result: auc.test.mean=0.9855,timetrain.test.mean=6.4926
##
```

c) Vizualize the results. Which learner would you use in practice and as a spam detector?

```
plotBMRBoxplots(bmr, measure = auc)
```




```
plotBMRBoxplots(bmr, measure = timetrain)
```



Tuning

Tune `mtry` and `nodesize` and `sampsize` of the random forest to get the best possible tuning error.

a) Define reasonable bounds for the parameter space. (Hint: Have a look at the number of rows and columns of the spam data)

b) Use a random search to optimize over the parameter space.

```
n = getTaskSize(spam.task)
p = getTaskNFeats(spam.task)
ps = makeParamSet(
    makeIntegerParam("mtry", lower = 1, upper = p),
    makeIntegerParam("nodesize", lower = 10, upper = 0.2 * n),
    makeIntegerParam("sampsize", lower = 1, upper = 0.6 * n)
)

tune.control = makeTuneControlRandom(maxit = 10)
lrn = makeLearner("classif.randomForest", predict.type = "prob")

res = tuneParams(lrn, spam.task, hout, auc, ps, tune.control)
```

```
## [Tune] Started tuning learner classif.randomForest for parameter set:
```

```
##             Type len Def        Constr Req Tunable Trafo
## mtry     integer   -   -       1 to 57   -    TRUE     -
## nodesize integer   -   -     10 to 920   -    TRUE     -
## sampsize integer   -   - 1 to 2.76e+03   -    TRUE     -
```

```
## With control class: TuneControlRandom
## Imputation value: -0
## [Tune-x] 1: mtry=47; nodesize=788; sampsize=1426
## [Tune-y] 1: auc.test.mean=0.9144; time: 0.0 min
## [Tune-x] 2: mtry=8; nodesize=527; sampsize=1288
## [Tune-y] 2: auc.test.mean=0.9570; time: 0.0 min
## [Tune-x] 3: mtry=49; nodesize=445; sampsize=614
## [Tune-y] 3: auc.test.mean=0.9175; time: 0.0 min
## [Tune-x] 4: mtry=28; nodesize=825; sampsize=1772
## [Tune-y] 4: auc.test.mean=0.9318; time: 0.0 min
## [Tune-x] 5: mtry=20; nodesize=528; sampsize=401
## [Tune-y] 5: auc.test.mean=0.9372; time: 0.0 min
## [Tune-x] 6: mtry=52; nodesize=685; sampsize=908
## [Tune-y] 6: auc.test.mean=0.9071; time: 0.0 min
## [Tune-x] 7: mtry=43; nodesize=559; sampsize=1929
## [Tune-y] 7: auc.test.mean=0.9438; time: 0.1 min
## [Tune-x] 8: mtry=11; nodesize=53; sampsize=421
## [Tune-y] 8: auc.test.mean=0.9689; time: 0.0 min
## [Tune-x] 9: mtry=36; nodesize=277; sampsize=1513
## [Tune-y] 9: auc.test.mean=0.9558; time: 0.1 min
## [Tune-x] 10: mtry=24; nodesize=843; sampsize=1458
## [Tune-y] 10: auc.test.mean=0.9336; time: 0.0 min
## [Tune] Result: mtry=11; nodesize=53; sampsize=421 : auc.test.mean=0.9689
```