Outline

Sarah Musiol

25/01/2021

Benchmark Analysis on Survival SVM

Problem: Factor nor character variables can be processed in the training of the model.

Temporary solution: Delete these variables from the dataset. Idea: Create a design matrix out of the dataset

Packages needed for this analysis

```r
require(mlr3)
require(mlr3tuning)

require(survivalsvm)

require(paradox)

# For survivalsvm learner
require(remotes)
#remotes::install_github("mlr-org/mlr3extralearners")
require(mlr3extralearners)
#install_learners("surv.svm")

# For Visualization
require(mlr3viz)


require(progressr)
```

Get the data

```r
train_data <- readRDS("../Data/train_data.Rds")
```

Check if problem exists in plain survivalsvm function

```r
svm_reg <-
  survivalsvm(
    formula = Surv(time, status) ~ .,
    data = train_data$d1,
    type = "regression",
    gamma.mu = 0.1
  )
```

It seems to be working in survivalsvm

Create Tasks for each dataset

```
tasks <- list()

for(d in 1:length(train_data)) {
  tasks[[names(train_data[d])]] <- mlr3proba::TaskSurv$new(
                                              id = names(train_data[d]),
                                              backend = train_data[[d]],
                                              time = "time",
                                              event = "status"
                                            )
}
```

Problem: Factor/Character Variable

```
train_data$d1 <- subset(train_data$d1, select = -V4)

tasks <-
  mlr3proba::TaskSurv$new(
    id = "d1",
    backend = train_data$d1,
    time = "time",
    event = "status"
  )
```

Graphical summary of the tasks properties by plotting each task.

```
par(mfrow = c(2,4))


autoplot(tasks[[1]])
autoplot(tasks[[2]])
autoplot(tasks[[3]])
autoplot(tasks[[4]])
autoplot(tasks[[5]])
autoplot(tasks[[6]])
autoplot(tasks[[7]])
autoplot(tasks[[8]])
```

Setup Autotuner

Define a search algorithm: random search with 10 evaluations Does that make sense?

```
random_tuner = mlr3tuning::tnr("random_search")
terminator = mlr3tuning::trm("evals", n_evals = 10)
```

Define performance measure

```
c_index <- mlr3::msr("surv.cindex")
IBS <- mlr3::msr("surv.graf")
```

Resampling strategy: 5-fold CV

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

Autotuner for SVM Regression

Check for each kernel and each method? Or do I only need one solution for each dataset? Meaning, best method and best kernel for that dataset? My idea: Like in the paper, check best solution for each kernel and method and compare with boxplots

Define svm regression as mlr3 learner

```
svr_lin =
  mlr3::lrn("surv.svm",
            type = "regression",
            gamma.mu = 0.1,
            kernel = "lin_kernel")

svr_add =
  mlr3::lrn("surv.svm",
            type = "regression",
            gamma.mu = 0.1,
            kernel = "add_kernel")

svr_rbf =
  mlr3::lrn("surv.svm",
            type = "regression",
            gamma.mu = 0.1,
            kernel = "rbf_kernel")
```

Define Search Space

```
svr_lin$param_set

params_svr = paradox::ParamSet$new(
  list(
    ParamDbl$new("gamma.mu", lower = 0L, upper = 5L, default = 0.1),
    ParamDbl$new("margin", lower = 0L, upper = 1L)
  )
)
```

Initialize Auto Tuner

```
svr_lin_tuner = mlr3tuning::AutoTuner$new(
  learner      = svr_lin,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svr,
  tuner        = random_tuner,
  terminator   = terminator
)

svr_add_tuner = mlr3tuning::AutoTuner$new(
  learner      = svr_add,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svr,
  tuner        = random_tuner,
  terminator   = terminator
)
```

```
svr_rbf_tuner = mlr3tuning::AutoTuner$new(
  learner      = svr_rbf,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svr,
  tuner        = random_tuner,
  terminator   = terminator
)
```

Autotuner for SVM Ranking

Define svm Ranking as mlr3 learner

```
svm_rank_lin =
  mlr3::lrn("surv.svm",
            type = "vanbelle2",
            gamma.mu = 0.1,
            kernel = "lin_kernel",
            diff.meth = "makediff3")

svm_rank_add =
  mlr3::lrn("surv.svm",
            type = "vanbelle2",
            gamma.mu = 0.1,
            kernel = "add_kernel",
            diff.meth = "makediff3")

svm_rank_rbf =
  mlr3::lrn("surv.svm",
            type = "vanbelle2",
            gamma.mu = 0.1,
            kernel = "rbf_kernel",
            diff.meth = "makediff3")
```

Define Search Space

```
svm_rank_lin$param_set

params_svm_rank = paradox::ParamSet$new(
  list(
    ParamDbl$new("gamma.mu", lower = 0L, upper = 5L, default = 0.1),
    ParamDbl$new("margin", lower = 0L, upper = 1L)
  )
)
```

Initialize Auto Tuner

```
svm_rank_lin_tuner = mlr3tuning::AutoTuner$new(
  learner      = svm_rank_lin,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svm_rank,
  tuner        = random_tuner,
  terminator   = terminator
)
```

```r
svm_rank_add_tuner = mlr3tuning::AutoTuner$new(
  learner      = svm_rank_add,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svm_rank,
  tuner        = random_tuner,
  terminator   = terminator
)

svm_rank_rbf_tuner = mlr3tuning::AutoTuner$new(
  learner      = svm_rank_rbf,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svm_rank,
  tuner        = random_tuner,
  terminator   = terminator
)
```

Autotuner for SVM Hybrid

Define svm Hybrid as mlr3 learner

```r
svm_hybrid_lin =
  mlr3::lrn("surv.svm",
            type = "hybrid",
            gamma.mu = 0.1,
            kernel = "lin_kernel",
            diff.meth = "makediff3")

svm_hybrid_add =
  mlr3::lrn("surv.svm",
            type = "hybrid",
            gamma.mu = 0.1,
            kernel = "add_kernel",
            diff.meth = "makediff3")

svm_hybrid_rbf =
  mlr3::lrn("surv.svm",
            type = "hybrid",
            gamma.mu = 0.1,
            kernel = "rbf_kernel",
            diff.meth = "makediff3")
```

Define Search Space

```r
svm_hybrid_lin$param_set

params_svm_hybrid = paradox::ParamSet$new(
  list(
    ParamDbl$new("gamma.mu", lower = 0L, upper = 5L, default = 0.1),
    ParamDbl$new("margin", lower = 0L, upper = 1L)
  )
)
```

Initialize Auto Tuner

```
svm_hybrid_lin_tuner = mlr3tuning::AutoTuner$new(
  learner      = svm_hybrid_lin,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svm_hybrid,
  tuner        = random_tuner,
  terminator   = terminator
)

svm_hybrid_add_tuner = mlr3tuning::AutoTuner$new(
  learner      = svm_hybrid_add,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svm_hybrid,
  tuner        = random_tuner,
  terminator   = terminator
)

svm_hybrid_rbf_tuner = mlr3tuning::AutoTuner$new(
  learner      = svm_hybrid_rbf,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params_svm_hybrid,
  tuner        = random_tuner,
  terminator   = terminator
)
```

Setup 1 Autotuner

Define learner

```
svm =
  mlr3::lrn("surv.svm",
            type = "regression",
            gamma.mu = 0.1)
```

Define Paramater Space

```
params = paradox::ParamSet$new(
  list(
    ParamDbl$new("gamma.mu", lower = 0L, upper = 5L, default = 0.1),
    ParamDbl$new("margin", lower = 0L, upper = 1L),
    ParamFct$new("type", levels = c("regression", "vanbelle1", "hybrid")),
    ParamFct$new("kernel", levels = c("lin_kernel", "add_kernel", "rbf_kernel")),
    ParamFct$new("diff.meth", levels = c("makediff3"), default = "makediff3")
  )
)
```

Setup Autotuner

```
svm_tuner = mlr3tuning::AutoTuner$new(
  learner      = svm,
  resampling   = rsmp_tuner,
  measure      = c_index,
  search_space = params,
  tuner        = random_tuner,
  terminator   = terminator
)
```

Benchmark

Define list of learners

```
coxph <- mlr3::lrn("surv.coxph")
kaplan <- mlr3::lrn("surv.kaplan")

learners <- list(svr_lin_tuner, svr_add_tuner, svr_rbf_tuner,
                 svm_rank_lin_tuner, svm_rank_add_tuner, svm_rank_rbf_tuner,
                 svm_hybrid_lin_tuner, svm_hybrid_add_tuner, svm_hybrid_rbf_tuner,
                 coxph, kaplan)

learners <- list(svr_lin_tuner,
                 svm_rank_lin_tuner,
                 svm_hybrid_lin_tuner,
                 coxph, kaplan)

learners <- list(svm_tuner, cosph, kaplan)
```

Define a benchmark grid.

```
rsmp_benchmark <- mlr3::rsmp("cv", folds = 10)

grd = mlr3::benchmark_grid(
  task = tasks,
  learner = learners,
  resampling = rsmp_benchmark
)
```

Run the benchmark

```
  future::plan("multisession", workers = 3)
  progressr::with_progress(bmr <- benchmark(grd))
  saveRDS(bmr, "../Data/bmr_cv.RDS")
```

Results

Evaluate results

```
#bmr$aggregate(IBS)
bmr$aggregate(c_index)

mlr3viz::autoplot(bmr, measure = c_index)
```