Forecasting in the mlr Framework

Steve Bronder
May 20th, 2017
Goal: Make Forecasting Simple

"We need to stop teaching abstinence and start teaching safe statistics" - Hadley Wickham

- No framework for tuning forecasting models
- R has great ML Packages and Forecasting Packages
- Forecasting and ML fields are divided
Goal: Simple Answer

Why don’t we have both?
Example Data

- Weekly Percent Change in Apple Closing Price
- Training: December 19th, 1980 : March 29th, 2017 (9147 days)
- Holdout: March 30th, 2017 : April 20th, 2017 (15 days)
Creating a Forecasting Task

- Task: Keeps data and meta-data for ML task

```r
library(mlr)
aaplXtsTrain = data.frame(coredata(aaplXtsTrain),
   daily_dates = index(aaplXtsTrain))
aaplXtsTest = data.frame(coredata(aaplXtsTest),
   daily_dates = index(aaplXtsTest))
aaplTask = makeForecastRegrTask(
   id = "Weekly Percent Change in Apple Closing Price",
   data = aaplXtsTrain,
   target = "Close",
   date.col = "daily_dates",
   frequency = 5L)
```
Creating a Forecasting Task: Info

aaplTask

Task: Weekly Percent Change in Apple Closing Price
Type: fcregr
Target: Close
Observations: 9147
Dates:
  Start: 1980-12-19
  End:   2017-03-29
Frequency: 5
Features:
  numerics factors ordered
     0      0      0
Missings: FALSE
Has weights: FALSE
Has blocking: FALSE
Training a Forecasting Learner

# Make learner

garch.mod = makeLearner("fcreg.rgarch", model = "sGARCH",
    garchOrder = c(1, 1), distribution.model = "sged",
    armaOrder = c(2, 2), n.ahead = 15, predict.type = "quantile",
    solver = "hybrid")

# Train and predict

garch.train = train(garch.mod, aaplTask)
predAapl = predict(garch.train, newdata = aaplXtsTest)

# Evaluate performance

performance(predAapl, measures = list(mase, rmse), task = aaplTask, model = garch.train)

    mase       rmse
0.17633953 0.01452162
Prediction Plot
Tuning a Model: Param Set and Search

par_set = makeParamSet(
    makeDiscreteParam(id = "model",
        values = c("sGARCH", "csGARCH", "fGARCH")),
    makeDiscreteParam("submodel", values = c("GARCH","TGARCH","AVGARCH"),
        requires = quote(model == 'fGARCH')),
    makeIntegerVectorParam(id = "garchOrder", len = 2L,
        lower = 1, upper = 15),
    makeIntegerVectorParam(id = "armaOrder", len = 2L,
        lower = 1, upper = 15),
    makeLogicalParam(id = "include.mean"),
    makeLogicalParam(id = "archm"),
    makeDiscreteParam(id = "distribution.model",
        values = c("norm","std","jsu","sged")),
    makeDiscreteParam(id = "stationarity", c(0,1)),
    makeDiscreteParam(id = "scale", c(0,1))
)

ctrl = makeTuneControlIrace(budget = 250)
Tuning a Model: Resampling

- 21 growing window CV

```r
resampDesc = makeResampleDesc("GrowingCV",
  horizon = 15L, initial.window = 0.5,
  skip = 220)
```
Tuning a Model: Tuning

garch.mod = makeLearner("fcreegr.garch", n.ahead = 15, solver = 'hybrid')
library(parallelMap)
parallelStart("multicore", 7, level = "mlr.resample")
configureMlr(on.learner.error = "warn")
set.seed(1234)
garch.res = tuneParams(garch.mod, task = aaplTask,
    resampling = resampDesc, par.set = par_set,
    control = ctrl,
    measures = mase)

parallelStop()
garch.res

Tune result:
Op. pars: model=sGARCH; garchOrder=4,4; armaOrder=8,9; include.mean=FALSE; archm=FALSE; distribution.model=norm; stationarity=0; fixed.se=1
mase.test.mean=0.494
Tuning a Model: Forecasting

```r
library(mlr)

# Train Final Model
garch.final = setHyperPars(garch.mod, par.vals = garch.res$x)
garch.train = train(garch.final, aaplTask)

# Predict
garch.pred = predict(garch.train, newdata = aaplXtsTest)
performance(garch.pred, measures = list(mase, rmse), task = aaplTask, model = garch.train)

      mase      rmse
0.099914625 0.009431966
```
Tuning a Model: Plot

Predictions Vs. Truth with 95% Confidence Intervals

colour
- Lower Quantile
- response
- truth
- Upper Quantile

dates

truth

Apr 03  Apr 10  Apr 17
Tuned Vs. Untuned
Conclusion

- We have a unified framework for forecasting and machine learning
- Forecasting mlr branch [here](#)
- Thank you for listening!
- [sab2287@columbia.edu](mailto:sab2287@columbia.edu)
# Make Standard Regression Task
```r
aaplRegTask <- makeRegrTask(id = "Weekly Percent Change in Apple Closing Price",
                          data = aaplXtsTrain[, "Close", drop = FALSE],
                          target = "Close")
```

# Use AR(p,d) pre-processing
```r
date_col = aaplXtsTrain$daily_dates
aaplXtsTest$daily_dates <- NULL
aaplLagTask = createLagDiffFeatures(aaplRegTask,
                      lag = 1L:500L,
                      difference = 1L:50L,
                      difference.lag = 1L:50L,
                      na.pad = FALSE,
                      date.col = date_col)
```
ranger.lrn = makeLearner("regr.ranger", num.threads = 7)
par.set = makeParamSet(
    makeIntegerParam("num.trees", lower = 200, upper = 2500),
    makeLogicalParam("respect.unordered.factors"),
    makeIntegerParam("mtry", lower = 1, upper = 80),
    makeIntegerParam("min.node.size", lower = 1, upper = 10),
    makeDiscreteParam("splitrule", values = c("variance", "maxstat")),
    makeNumericParam("alpha", lower = .001, upper = .8 ),
    makeNumericParam("minprop", lower = .001, upper = .5)
)
library(mlrMBO)

mbo.ctrl = makeMBOControl()

mbo.ctrl = setMBOControlTermination(mbo.ctrl,
  max.eval = 100)

ctrl = mlr:::makeTuneControlMBO(mbo.control = mbo.ctrl)

tune_mod <- tuneParams(learner = ranger.lrn,
  task = aaplLagTask, measures = mase,
  resampling = resampDesc, par.set = par.set,
  control = ctrl)

tune_mod

Tune result:
Op. pars: num.trees=1295; respect.unordered.factors=TRUE; mtry=79; min.node.size=2; splitrule=
mase.test.mean=0.31
Forecasting with ML: Forecast

ranger.final = setHyperPars(ranger.learner, 
    par.vals = tune_mod$x)

ranger.trn.model = train(ranger.final, aaplLagTask)

ranger.fc = forecast(ranger.trn.model, h = 15, 
    newdata = aaplXtsTest[, "Close", drop = FALSE])

performance(ranger.fc, list(mase, rmse), 
    task = aaplLagTask, model = ranger.trn.model)

    mase       rmse
0.44726774  0.01561785
Forecasting with ML: Plot