Adventures in Supercomputing with R

Lecture 4: Shared Memory (continued)

George Ostrouchov

Oak Ridge National Laboratory and University of Tennessee

2022/02/28 (updated: 2022-03-09)

Questions and Review

- RStudio - GitHub - It4I workflow?
- Are you using another workflow?
- IT4I experience?

- Measurement and terminology of parallel speedup
- Unix fork and its use by `mclapply()`
- Random Forest Classification

Resampling … and Crossvalidation

More shared memory parallelization, and a preview of parallel linear algebra …

Bootstrap, Bagging, Boosting, and Crossvalidation

- **Bootstrap**: a tool for assessing statistical accuracy
  - Resampling data with replacement and repeating estimation
  - Results in a sample of parameter estimates
- **Bagging** (bootstrap aggregation): a tool for reducing the variance of a prediction function
  - Simple models (low bias and high variance models) on resampled data
  - Consensus prediction (majority vote or average)
  - Generalized by **random forest** to sampling subsets of predictors
- **Boosting**: forward additive modeling, a greedy method of growing a model
  - Introduced via increasing weights on misclassified observations but shown* to fit additive model framework
  - Sequential, so parallelization within a model
- **Crossvalidation**: model performance assessment
  - Estimates expected prediction error
  - Uses all data (no test set)

*Hastie, Tibshirani, and Friedman (2009) The Elements of Statistical Learning, Second Edition, (2009). link

Bootstrap

- Data: $\mathbf{Z} = (z_1, z_2, \ldots, z_N)$, where $z_i = (y_i, x_i)$
- Model: Let $S(\mathbf{Z})$ be an estimated quantity from the data
- Sample with replacement B sets of size N from data
- Fit model to each of the reseampled B sets $\{S(\mathbf{Z}^{*1}), S(\mathbf{Z}^{*2}), \ldots, S(\mathbf{Z}^{*B})\}$
- Use as sample from the sampling distribution of the estimator
- For example, $\widehat{\mathrm{Var}[S(\mathbf{Z})]} = \frac{1}{B-1} \sum_{b=1}^{B} [S(\mathbf{Z}^{*b}) - \bar{S}^*]^2$

Easy to parallelize over the B sets

Further opportunities may exist within the estimator $S(\cdot)$

Bagging (bootstrap aggregation)

- Let $\widehat{S(\mathbf{Z})} = \frac{1}{B}\sum_{b=1}^{B} S(\mathbf{Z}^{*b})$
- Majority vote if discrete
- Reduces variability of the estimate

Easy to parallelize over B

Further opportunities may exist within the estimator $S(\cdot)$

Random Forest for Regression or Classification

1. For b = 1 to B:
    - Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
    - Grow a random-forest tree T_b on $\mathbf{Z}^*$:
      Recursively, for each terminal node, until n_{min} node size:
        - Select m variables at random from the p variables
        - Pick the best variable/split-point among the m
        - Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at x:

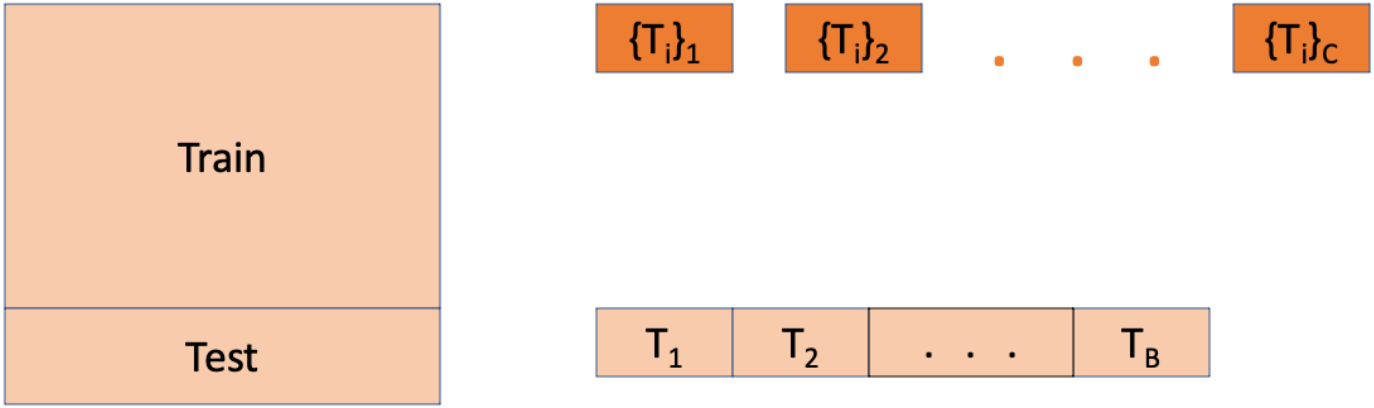
Regression: $\hat{f}_{\mathrm{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree.

Then $\hat{C}_{\mathrm{rf}}^B(x)$ = majority vote $\{\hat{C}_b(x)\}_1^B$.

*Algorithm 15.1 in Hastie, Tibshirani, and Friedman (2009). The Elements of Statistical Learning, Second Edition. Link

Shared memory considerations



Cores produce separate forests:

Combine forests for prediction or combine predictions?

Boosting

Discrete AdaBoost *

1. Initialize weights $w_i = \frac{1}{N}, i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
    - Fit a classifier $G_m(x)$ to the training data using weights w_i.
    - Compute

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1} N w_i}$$

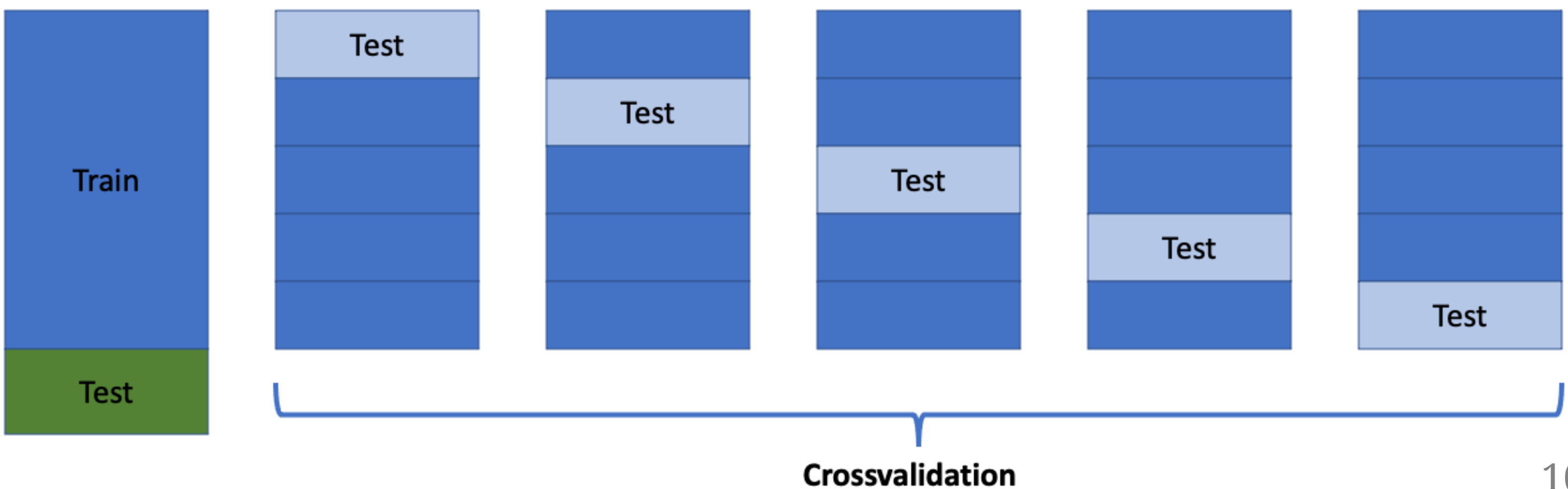
    - Compute $\alpha_m = \log((1 - err_m)/err_m)$.
    - Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, 2, \ldots, N$.
3. Output $G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Sequential over M, parallelize within $G_m(\cdot)$

* Algorithm 10.1 in Hastie, Tibshirani, and Friedman (2009)

K-fold Crossvalidation

- Randomly divide data into k roughly equal folds
- Let $\hat{f}^{-k}$ be the estimator when fold k is removed
- let k_i be the assigned fold of observation i
- Let $f(x, \alpha)$ be an estimator with a tuning parameter α
- Then $\mathrm{CV}(f, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-k_i}(x_i, \alpha))$ is the crossvalidation estimate of the prediction error



KPMS-IT4I-EX/code/rf_serial.r

```
library(randomForest)
data(LetterRecognition, package = "mlbench")
set.seed(seed = 123)

n = nrow(LetterRecognition)
n_test = floor(0.2 * n)
i_test = sample.int(n, n_test)
train = LetterRecognition[-i_test, ]
test = LetterRecognition[i_test, ]

rf.all = randomForest(lettr ~ ., train, ntree = 500, norm.votes = FAL
pred = predict(rf.all, test)

correct = sum(pred == test$lettr)
cat("Proportion Correct:", correct/(n_test), "\n")

plot(rf.all)
```

KPMS-IT4I-EX/code/rf_cv_serial.r

```r
set.seed(seed = 123)

ntree = 300
nfolds = 10
mtry_val = 1:(ncol(train) - 1)
folds = sample( rep_len(1:nfolds, nrow(train)), nrow(train) )
cv_df = data.frame(mtry = mtry_val, incorrect = rep(0, length(mtry_va
cv_pars = expand.grid(mtry = mtry_val, f = 1:nfolds)
fold_err = function(i, cv_pars, folds, train) {
  mtry = cv_pars[i, "mtry"]
  fold = (folds == cv_pars[i, "f"])
  rf.all = randomForest(lettr ~ ., train[!fold, ], ntree = ntree,
                        mtry = mtry, norm.votes = FALSE)
  pred = predict(rf.all, train[fold, ])
  sum(pred != train$lettr[fold])
}

cat("Running serial\n")
system.time({
cv_err = lapply(1:nrow(cv_pars), fold_err, cv_pars, folds = folds, tr
cv_err = tapply(unlist(cv_err), cv_pars[, "mtry"], sum)
})
png(paste0("rf_cv_mc0.png")); plot(mtry_val, cv_err/(n - n_test)); de
```

KPMS-IT4I-EX/code/rf_cv_mc.r

```r
set.seed(seed = 123, "L'Ecuyer-CMRG")

ntree = 200
nfolds = 10
mtry_val = 1:(ncol(train) - 1)
folds = sample( rep_len(1:nfolds, nrow(train)), nrow(train) )
cv_df = data.frame(mtry = mtry_val, incorrect = rep(0, length(mtry_va
cv_pars = expand.grid(mtry = mtry_val, f = 1:nfolds)
fold_err = function(i, cv_pars, folds, train) {
  mtry = cv_pars[i, "mtry"]
  fold = (folds == cv_pars[i, "f"])
  rf.all = randomForest(lettr ~ ., train[!fold, ], ntree = ntree,
                        mtry = mtry, norm.votes = FALSE)
  pred = predict(rf.all, train[fold, ])
  sum(pred != train$lettr[fold])
}

nc = as.numeric(commandArgs(TRUE)[1])
cat("Running with", nc, "cores\n")
system.time({
cv_err = parallel::mclapply(1:nrow(cv_pars), fold_err, cv_pars, folds
                              train = train, mc.cores = nc)
err = tapply(unlist(cv_err), cv_pars[, "mtry"], sum)
})
```

Demo …

Matrix Libraries ...

R-LAPACK-BLAS

- BLAS: Basic Linear Algebra Subroutines - A matrix multiplication library
    - vector-vector (Level-1), matrix-vector (Level-2), matrix-matrix (Level-3)
- LAPACK: dense and banded matrix decompositions and more
    - $LU \quad LL^T \quad QR \quad UDV^T \quad VD^2V^T \quad \|\cdot\|_p$
- Implementations: OpenBLAS, Intel MKL, Nvidia nvBLAS, Apple vecLib, AMD BLIS, Arm Performance Libraries
- **FlexiBLAS**: A BLAS and LAPACK wrapper library with runtime exchangable backends

FlexiBLAS

```r
library(flexiblas)

# check whether FlexiBLAS is available
flexiblas_avail()
#> [1] TRUE

# get the current backend
flexiblas_current_backend()
#> [1] "OPENBLAS-OPENMP"

# list all available backends
flexiblas_list()
#> [1] "NETLIB"           "__FALLBACK__"     "BLIS-THREADS"     "OPE
#> [5] "BLIS-SERIAL"      "ATLAS"            "OPENBLAS-SERIAL"  "OPE
#> [9] "BLIS-OPENMP"

# get/set the number of threads
flexiblas_set_num_threads(12)
flexiblas_get_num_threads()
#> [1] 12
```

https://github.com/Enchufa2/r-flexiblas
https://cran.r-project.org/package=flexiblas