Risk Estimation

-Quantifying the World-

Lecturer: Darren Homrighausen, PhD

Loss functions and risk

Define a function[1] $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that smaller values of ℓ indicate better performance

Two important examples:

- $\ell(\hat{f}(X), Y) = (\hat{f}(X) - Y)^2$ (regression, square-error)
- $\ell(\hat{f}(X), Y) = \mathbf{1}(\hat{f}(X) \neq Y)$ (classification, 0-1)

These expressions are both random variables.

This leads us to define the prediction or estimation risk of a procedure $\hat{f}$ to be

$$R(\hat{f}) = \mathbb{E}\ell(\hat{f}(X), Y)$$

[1] This is the loss for prediction. Other tasks, such as estimation, may have a different domain.

Risk estimation

The prediction risk can be written

$$R(f) = \mathbb{E}\ell(f(X), Y) \leftrightarrow \text{Bias} + \text{Variance}$$

The overriding theme is that we would like to add a judicious amount of bias to get lower risk

As R isn't known, we need to estimate it

From the Preliminary Materials, $\hat{R} = \frac{1}{n}\sum_{i=1}^{n} \ell(f(X_i), Y_i)$ isn't very good

(In fact, one tends to not add bias when estimating R with $\hat{R}$)

Risk estimation: a general form

The problem is that $\hat{R}$ is overly optimistic

The average optimism is

$$\mathrm{opt} =^{*} \mathbb{E}[R - \hat{R}]$$

Typically, opt is positive as $\hat{R}$ will underestimate the risk

($*$ See Elements of Statistical Learning, Chapter 7 for details for a more precise statement)

Risk estimation: a general form

It turns out for a variety of ℓ (such as squared error and 0-1)

$$\text{opt} = \frac{2}{n}\sum_{i=1}^{n} \text{Cov}(\hat{f}(X_i), Y_i)$$

This is related intimately with degrees of freedom

$$\text{df} = \frac{1}{\sigma^2}\sum_{i=1}^{n} \text{Cov}(\hat{f}(X_i), Y_i) = \frac{n}{2\sigma^2}\text{opt}$$

$(\sigma^2 = \mathbb{V}Y_i)$

Example: For multiple regression (i.e. $\hat{f}(X) = \hat{\beta}_{LS}^\top X$),

$$\text{df} = \text{trace}(\mathbb{X}(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top) = \text{rank}(\mathbb{X})$$

A risk estimate

Therefore, we get the following expression of risk

$$\mathrm{GIC} = \hat{R} + \widehat{\mathrm{opt}}$$

(Writing GIC indicates generalized information criterion)

Differing $\widehat{\mathrm{opt}}$ leads to AIC, BIC, Mallows Cp, and others

$\widehat{\mathrm{opt}}$ depends on:

- a variance estimator $\hat{\sigma}$
- a scaling term

Various forms of risk estimates

$$\mathrm{AIC} = \hat{R} + 2 \cdot \mathrm{df} \cdot \hat{\sigma}^2 / n$$

$$\mathrm{AICc}(\hat{\beta}) = \mathrm{AIC} + \frac{2\mathrm{df} \cdot (\mathrm{df} + 1)}{n - \mathrm{df} - 1}$$

$$\mathrm{BIC} = \hat{R} + \log(n) \cdot \mathrm{df} \cdot \hat{\sigma}^2 / n$$

Including more parameters leads to:

- a smaller $\hat{R}$
- a larger $\widehat{\mathrm{opt}}$

Goal: Now, we can use one of the GIC procedures to tell us which model to use

(As long as $\log n \geq 2$, BIC picks a smaller model than AIC)

Cross-validation

A different approach to risk estimation

Let (X_0, Y_0) be a test observation, identically distributed as an element in $\mathcal{D}$, but also independent of $\mathcal{D}$.

$$R(f) = \mathbb{E}\ell(f(X_0), Y_0) \underbrace{=}_{\text{regression}} \mathbb{E}(Y_0 - f(X_0))^2$$

Of course, the quantity $(Y_0 - f(X_0))^2$ is an unbiased estimator of $R(f)$ and hence we could use it to estimate $R(f)$

However, we don't have any such new observation

Or do we?

An intuitive idea

Let's set aside one observation and predict it

For example: Set aside (X_1, Y_1) and fit $\hat{f}^{(1)}$ on $(X_2, Y_2), \ldots, (X_n, Y_n)$

(The notation $\hat{f}^{(1)}$ just symbolizes leaving out the first observation before fitting $\hat{f}$)

$$R_1(\hat{f}^{(1)}) = (Y_1 - \hat{f}^{(1)}(X_1))^2$$

As the left off data point is independent of the data points used for estimation,

$$\mathbb{E}R_1(\hat{f}^{(1)}) \approx R(\hat{f})$$

Leave-one-out cross-validation

Cycling over all observations and taking the average produces
leave-one-out cross-validation

$$\mathrm{CV}_n(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} R_i(\hat{f}^{(i)}) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{f}^{(i)}(X_i))^2.$$

More general cross-validation schemes

Let $\mathcal{N} = \{1, \ldots, n\}$ be the index set for $\mathcal{D}$

- K-fold: Fix $V = \{v_1, \ldots, v_K\}$ such that $v_j \cap v_k = \emptyset$ and $\bigcup_j v_j = \mathcal{N}$

$$\mathrm{CV}_K(\hat{f}) = \frac{1}{K} \sum_{v \in V} \frac{1}{|v|} \sum_{i \in v} (Y_i - \hat{f}^{(v)}(X_i))^2$$

- Bootstrap: Instead of partitioning, we could make K bootstrap draws and average
- Factorial: We could make all subsets of $\mathcal{N}$ and average

More general cross-validation schemes: A comparison

- CV_K gets more computationally demanding as $K \to n$
- The bias of CV_K goes down, but the variance increases as $K \to n$
- The factorial version isn't commonly used

  (Very computationally demanding)

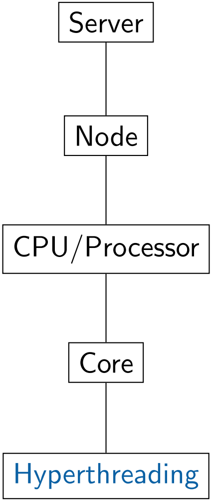
Summary time

CV	Prediction risk consistent. Generally selects a model larger than necessary
AIC	Provides optimal risk estimation (and is asymptotically equivalent to CV). Inconsistent for model selection
BIC	Consistent for model selection consistent, sub optimal for risk estimation

Aside: There exist impossibility theorems stating that risk estimation procedures good at prediction are bad at model selection (and vice-versa)

Parallelism

Distributed computing hierarchy



Example: A server might have

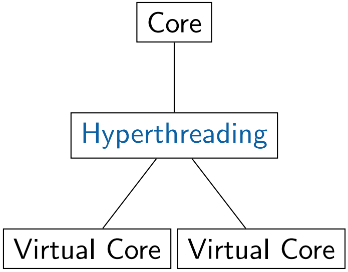
- 64 nodes
- 2 processors per node
- 16 cores per processor
- hyper threading

The goal is to somehow allocate a job so that these resources are used efficiently

Jobs are composed of threads, which are specific computations

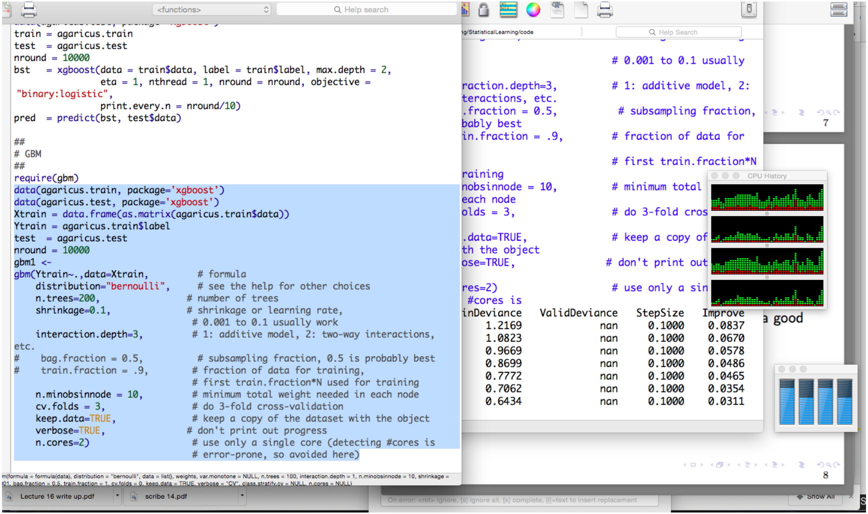
Hyperthreading

Developed by Intel, Hypertheading allows for each core to pretend to be two cores



This works by trading off computation and read-time for each core

GBM: Figures



train = agaricus.train
test = agaricus.test
nround = 10000
bst = xgboost(data = train$data, label = train$label, max.depth = 2,
eta = 1, nthread = 1, nround = nround, objective =
"binary:logistic",
print.every.n = nround/10)
pred = predict(bst, test$data)
##
GBM
##
require(gbm)
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
Xtrain = data.frame(as.matrix(agaricus.train$data))
Ytrain = agaricus.train$label
test = agaricus.test
nround = 10000
gbm1 <-
gbm(Ytrain~.,data=Xtrain, # formula
distribution="bernoulli", # see the help for other choices
n.trees=200, # number of trees
shrinkage=0.1, # shrinkage or learning rate,
# 0.001 to 0.1 usually work
interaction.depth=3, # 1: additive model, 2: two-way interactions,
etc.
# bag.fraction = 0.5, # subsampling fraction, 0.5 is probably best
# train.fraction = .9, # fraction of data for training,
# first train.fraction*N used for training
n.minobsinnode = 10, # minimum total weight needed in each node
cv.folds = 3, # do 3-fold cross-validation
keep.data=TRUE, # keep a copy of the dataset with the object
verbose=TRUE, # don't print out progress
n.cores=2) # use only a single core (detecting #cores is
# error-prone, so avoided here)
inDeviance ValidDeviance StepSize Improve
1.2169 nan 0.1000 0.0837
1.0823 nan 0.1000 0.0670
0.9669 nan 0.1000 0.0578
0.8699 nan 0.1000 0.0486
0.7772 nan 0.1000 0.0465
0.7062 nan 0.1000 0.0354
0.6434 nan 0.1000 0.0311
CPU History
Lecture 16 write up.pdf
scribe 14.pdf

Parallelism for CV in R

Go to `crossValidationParallel.R`