Machine Learning: II

Zhentao Shi

March 14, 2019

Machine Learning II

In this lecture, we introduce supervised learning methods that induces data-driven interaction of the covariates. The interaction makes the covariates much more flexible to capture the subtle feature in the data. However, insufficient theoretical understanding is shed light on these methods due to the complex nature, so they are often viewed by theorists as "black-boxes" methods. In real applications, when the machines are carefully tuned, they can achieve surprisingly superior performance. Gu, Kelly, and Xiu (2018) showcases a horse race of a myriad of methods, and the general message is that interaction helps with forecast in the financial market. In the meantime, industry insiders are pondering whether these methods are "alchemy" that fall short of scientific standard. Extreme caution must be exercised when we apply these methods in empirical economic analysis.

Regression Tree and Bagging

We consider supervised learning setting in which we use x to predict y. It can be done by traditional nonparametric methods such as kernel regression. Regression tree (Breiman et al. 1984) is an alternative method. It recursively partitions the the space of the regressors. The algorithm is easy to describe: each time a covariate is split into two dummies, and the splitting criterion is aggressive reduction of the SSR. In the formulate of the SSR, the fitted value is computed as the average of y_i's in a partition.

The tuning parameter is the depth of the tree, which is referred to the number of splits. Given a dataset d and the depth of the tree, the fitted regression tree $\hat{r}(d)$ is completely determined by the data.

- Example: Using longitude and latitude for Beijing Housing price.

The problem of the regression tree is its instability. For data generated from the data DGP, the covariate chosen to be split and the splitting point can be vary widely and they heavily influence the path of the partition.

Bootstrap averaging, or *bagging* for short, was introduced to reduce the variance of the regression tree (Breiman 1996). Bagging grows a regression tree for each bootstrap sample, and then do a simple average. Let $d^{\star b}$ be the b-th bootstrap sample of the original data d, and then the bagging estimator is defined as

$$\hat{r}_{\text{bagging}} = B^{-1} \sum_{b=1}^{B} \hat{r}(d^{\star b}).$$

It is an example of the *ensemble learning*.

Inoue and Kilian (2008) is an early application of bagging in time series forecast. Hirano and Wright (2017) provide a theoretical perspective on the risk reduction of bagging.

Random Forest

Random forest (Breiman 2001) shakes up the regressors by randomly sampling m out of the total p covarites before *each split of a tree*. The tuning parameters in random forest is the tree depth and m. Due to the "de-correlation" in sampling the regressors, it in general performs better than bagging in prediction exercises.

Below is a very simple real data example of random forest using the Boston Housing data.

```
require(randomForest)
```

```
## Loading required package: randomForest
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

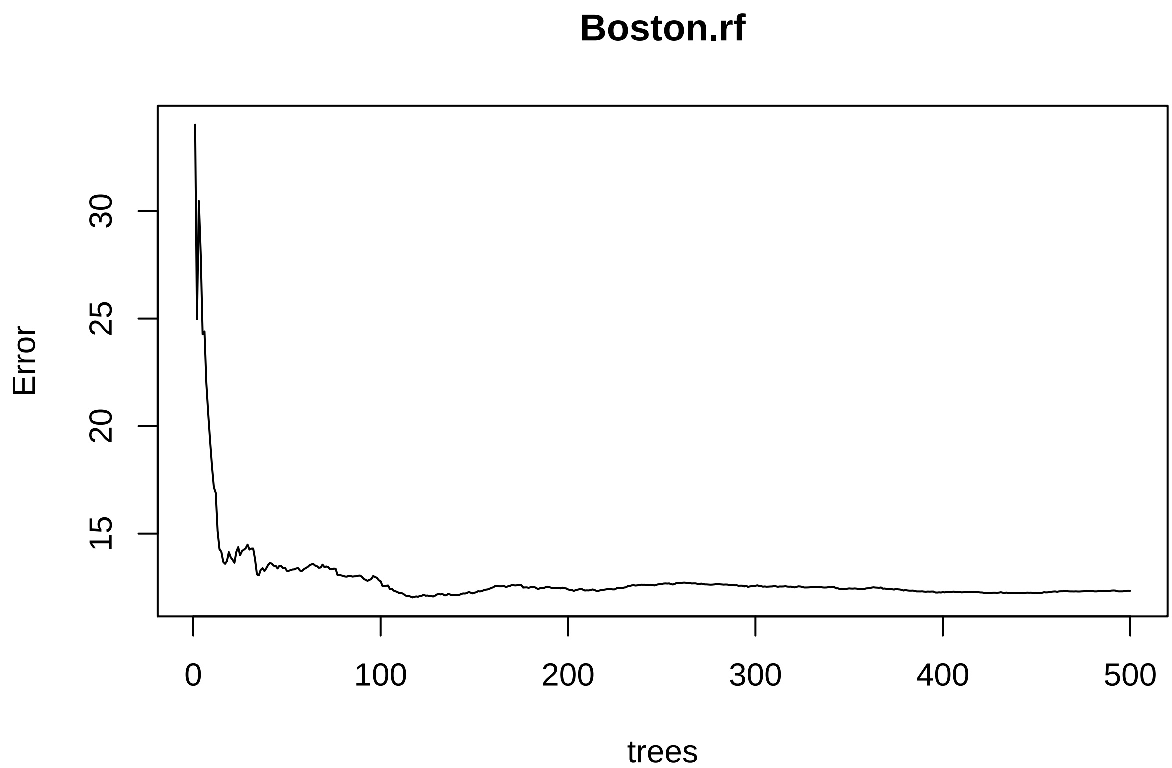
```
require(MASS)#Package which contains the Boston housing dataset
```

```
## Loading required package: MASS
## Warning: package 'MASS' was built under R version 3.4.2
```

```
attach(Boston)
set.seed(101)

#training Sample with 300 observations
train=sample(1:nrow(Boston),300)

Boston.rf=randomForest(medv ~ . , data = Boston , subset = train)
plot(Boston.rf)
```




```r
# getTree(Boston.rf)
importance(Boston.rf)
```

```
##         IncNodePurity
## crim        1487.1777
## zn           142.0280
## indus        965.7756
## chas         234.6918
## nox         1741.9305
## rm          7435.3378
## age          655.6031
## dis         1357.3411
## rad          316.3278
## tax          794.0953
## ptratio     1858.7183
## black        455.5382
## lstat       6947.9121
```

Despite the simplicity of the algorithm, the consistency of random forest is not proved until Scornet, Biau, and Vert (2015), and inferential theory was first established by Wager and Athey (2018) in the context of treatment effect estimation. Athey, Tibshirani, and Wager (2019) generalizes CART to local maximum likelihood.

Gradient Boosting

Bagging and random forest almost always use equal weight on each generated tree for the ensemble. Tree boosting takes a distinctive scheme to determine the ensemble weights. It is a deterministic approach that does not resample the original data.

1. Use the original data $d^0 = (x_i, y_i)$ to grow a shallow tree $\hat{r}^0(d^0)$. Save the prediction $f_i^0 = \alpha \cdot \hat{r}^0(d^0, x_i)$ where $\alpha \in [0,1]$ is a shrinkage tuning parameter. Save the residual $e_i^0 = y_i - f_i^0$. Set $m = 1$.
2. In the m-th iteration, use the data $d^m = (x_i, e_i^{m-1})$ to grow a shallow tree $\hat{r}^m(d^m)$. Save the prediction $f_i^m = f_i^{m-1} + \alpha \cdot \hat{r}^m(d, x_i)$. Save the residual $e_i^m = y_i - f_i^m$. Update $m = m + 1$.
3. Repeat Step 2 until $m > M$.

In this boosting algorithm there are three tuning parameters: the tree depth, the shrinkage level α, and the number of iterations M. The algorithm can be sensitive to all the three tuning parameters. When a model is tuned well, it often performs remarkably. For example, the script `Beijing_housing_gbm.R` achieves 92% out-of-sample R^2 (in comparison with OLS's 84%) in the Beijing housing data. This script implements boosting via the package `gbm`, which stands for "Gradient Boosting Machine."

There are many variants of boosting algorithms. For example, L_2-boosting, componentwise boosting, and AdaBoosting, etc. Statisticians view boosting as a gradient descent algorithm to reduce the risk. The fitted tree in each iteration is the deepest descent direction, while the shrinkage tames the fitting to avoid proceeding too aggressively.

- Shi (2016) proposes a greedy algorithm in similar spirit to boosting for moment selection in GMM.
- Phillips and Shi (2018, to be released) uses L_2-boosting as a boosted version of the Hodrick-Prescott filter.
- Shi and Huang (2018, to be released)

Neural Network

A neural network is the workhorse behind Alpha-Go and self-driven cars. However, from a statistician's point of view it is just a particular type of nonlinear models. Figure 1 illustrates a one-layer neural network, but in general there can be several layers. The transition from layer $k-1$ to layer k can be written as

$$
\begin{aligned}
z_l^{(k)} &= w_{l0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} w_{lj}^{(k-1)} a_j^{(k-1)} \qquad (1) \\
a_l^{(k)} &= g^{(k)}(z_l^{(k)}),
\end{aligned}
$$

where $x_j = a_j^{(0)}$ is the input. The above formulation shows that $z_l^{(k)}$ usually takes a linear form, while the *activation function* $g(\cdot)$ can be an identify function or a simple nonlinear function. Popular choices of the activation function are sigmoid $(1/(1+\exp(-x)))$ and rectified linear unit (ReLu, $z \cdot 1\{x \geq 0\}$), etc.

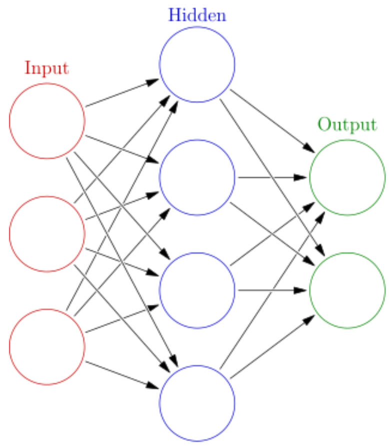


Figure 1: A Single Layer Feedforward Neural Network

A user has several decisions to make when fitting a neural network: besides the activation function, the tuning parameters are the number of hidden layers and the number of nodes in each layer. Many free parameters are generated from the multiple layer and multiple nodes, and in estimation often regularization methods are employed to penalize the l_1 and/or l_2 norms, which requires extra tuning parameters. `data_example/Keras_ANN.R` gives an example of a neural network with two hidden layers, each has 64 nodes, and the ReLu activation function.

Due to the nonlinear nature of the neural networks, theoretical understanding about its behavior is still scant. One of the early contributions came from econometrician: Hornik, Stinchcombe, and White (1989) (Theorem 2.2) show that a single hidden layer neural network, given enough many nodes, is a *universal approximator* for any measurable function.

After setting up a neural network, the free parameters must be determined by numerical optimization. The nonlinear complex structure makes the optimization very challenging and the global optimizer is beyond guarantee. In particular, when the sample size is big, the de facto optimization algorithm is the stochastic gradient descent.

Thanks to computational scientists, Google's `tensorflow` is perhaps the most popular backend of neural network estimation, and `keras` is the deep learning modeling language. Their relationship is similar to `Mosek` and `CVXR`.

Stochastic Gradient Descent (SGD)

In optimization we update the parameter

$$\beta_{k+1} = \beta_k + a_k p_k,$$

where $a_k \in \mathbb{R}$ is the step length and p_k is a vector of directions. Use a Talyor expansion,

$$f(\beta_{k+1}) = f(\beta_k + a_k p_k) \approx f(\beta_k) + a_k \nabla f(\beta_k) p_k,$$

If in each step we want the value of the criterion function $f(x)$ to decrease, we need $\nabla f(\beta_k) p_k \leq 0$. A simple choice is $p_k = -\nabla f(\beta_k)$, which is called the deepest decent. Newton's method corresponds to $p_k = -(\nabla^2 f(\beta_k))^{-1} \nabla f(\beta_k)$, and BFGS uses a low-rank matrix to approximate $\nabla^2 f(\beta_k)$.

When the sample size is huge and the number of parameters is also big, the evaluation of the gradient can be prohibitively expensive. SGD uses a small batch of the sample to evaluate the gradient in each iteration. It can significantly save computational time. It is the de facto optimization procedure in complex optimization problems such as training a neural network.

However, SGD involves tuning parameters (say, the batch size and the learning rate) that can dramatically affect the outcome, in particular in nonlinear problems. Careful experiments must be carried out before serious implementation.

Below is an example of SGD in the PPMLE with sample size 100,000 and the number of parameters 100. SGD is usually much faster.

The new functions are defined with the data explicity as arguments. Because in SGD each time the log-likelihood function and the gradiant are evaluated at a different subsample.

```
poisson.loglik = function( b, y, X ) {
  b = as.matrix( b )
  lambda =  exp( X %*% b )
  ell = -mean( -lambda + y *  log(lambda) )
  return(ell)
}


poisson.loglik.grad = function( b, y, X ) {
  b = as.matrix( b )
  lambda =  as.vector( exp( X %*% b ) )
  ell = -colMeans( -lambda * X + y * X )
  ell_eta = ell
  return(ell_eta)
}
```

```
##### generate the artificial data
set.seed(898)
nn = 1e+05
K = 100
X = cbind(1, matrix(runif(nn * (K - 1)), ncol = K - 1))
b0 = rep(1, K)/K
y = rpois(nn, exp(X %*% b0))


b.init = runif(K)
b.init = 2 * b.init/sum(b.init)

# and these tuning parameters are related to N and K

n = length(y)
test_ind = sample(1:n, round(0.2 * n))

y_test = y[test_ind]
X_test = X[test_ind, ]
```

```r
y_train = y[-test_ind]
X_train = X[-test_ind, ]
```

```r
# optimization parameters

# sgd depends on * eta: the learning rate * epoch: the averaging small batch
# * the initial value

set.seed(105)

max_iter = 5000
min_iter = 20
eta = 0.01
epoch = round(100 * sqrt(K))


b_old = b.init

pts0 = Sys.time()
# the iteration of gradient
for (i in 1:max_iter) {

    loglik_old = poisson.loglik(b_old, y_train, X_train)
    i_sample = sample(1:length(y_train), epoch, replace = TRUE)
    b_new = b_old - eta * poisson.loglik.grad(b_old, y_train[i_sample], X_train[i_sample,
        ])
    loglik_new = poisson.loglik(b_new, y_test, X_test)
    b_old = b_new  # update

    criterion = loglik_old - loglik_new
    # cat( 'the ', i, '-th criterion = ', criterion, '\n')

    if (criterion < 1e-04 & i >= min_iter)
        break
}
cat("point estimate =", b_new, ", log_lik = ", loglik_new, "\n")
```

```
## point estimate = -0.007249861 0.0138033 0.006121163 0.006204153 0.001798774 -0.009304036 -0
```

```r
pts1 = Sys.time() - pts0
print(pts1)
```

```
## Time difference of 0.623333 secs
```

```r
# optimx is too slow for this dataset.  Nelder-Mead method is too slow for
# this dataset

# thus we only sgd with NLoptr
```

```
opts = list(algorithm = "NLOPT_LD_SLSQP", xtol_rel = 1e-07, maxeval = 5000)


pts0 = Sys.time()
res_BFGS = nloptr::nloptr(x0 = b.init, eval_f = poisson.loglik, eval_grad_f = poisson.loglik.gr
    opts = opts, y = y_train, X = X_train)
print(res_BFGS)
```

```
##
## Call:
##
## nloptr::nloptr(x0 = b.init, eval_f = poisson.loglik, eval_grad_f = poisson.loglik.grad,
##     opts = opts, y = y_train, X = X_train)
##
##
## Minimization using NLopt version 2.4.2
##
## NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because
## xtol_rel or xtol_abs (above) was reached. )
##
## Number of Iterations....: 50
## Termination conditions:  xtol_rel: 1e-07 maxeval: 5000
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  0.817239347882672
## Optimal value of controls: 0.008377464 0.00223468 0.006802827 0.009527097 -0.009179097 0.00
## -0.001861563 0.0250962 0.007914374 0.01430903 0.0185239 0.0009192213
## -0.003079815 0.02091482 0.01838836 0.02317564 0.007484981 0.02760165
## 0.02395294 0.02581524 0.0008703182 0.003018651 -0.0008777799 0.005505568
## -0.006333724 0.007045993 0.009869454 0.01635177 0.02444754 0.01859429
## 0.002282465 -0.006204774 0.01787497 0.00871074 0.01634583 0.01452785
## 0.01535949 0.02114798 0.02882988 0.003990018 0.006488755 0.01821596
## 0.01224189 -0.001992572 0.009518121 -0.004068174 -0.00955518 -0.008724836
## 0.01379455 0.01182887 0.02293765 0.01606992 0.01559747 0.02075959
## -0.002392985 -0.003346411 0.006898647 0.0120779 0.02070444 0.01023695
## 0.01356451 0.01390674 0.01992954 0.00914424 0.01823229 0.009599003
## 0.005063411 0.001639757 -0.003360399 0.004146783 0.0004618288 0.01613158
## 0.00628123 -0.007416353 0.01045353 0.02667767 0.01690433 0.009531548
## 0.01430839 0.005220393 0.02352873 0.01921597 0.01270081 0.00527763
## 0.008045643 0.002385956 0.02469504 0.002604459 0.02668898 -0.01424429
## 0.01709918 0.008055604 0.007352139 0.01098655 0.002287008 0.0190659
## 0.0004742784 0.01790932 -0.0007725258 0.02297663
```

```
pts1 = Sys.time() - pts0
print(pts1)
```

```
## Time difference of 6.658961 secs
```

```r
b_hat_nlopt = res_BFGS$solution


#### evaluation in the test sample
cat("\n\n\n\n\n\n\n\n")
```

```r
cat("log lik in test data by sgd = ", poisson.loglik(b_new, y = y_test, X_test),
    "\n")
```

```
## log lik in test data by sgd =  0.8261549
```

```r
cat("log lik in test data by nlopt = ", poisson.loglik(b_hat_nlopt, y = y_test,
    X_test), "\n")
```

```
## log lik in test data by nlopt =  0.8263522
```

```r
cat("log lik in test data by oracle = ", poisson.loglik(b0, y = y_test, X_test),
    "\n")
```

```
## log lik in test data by oracle =  0.8254042
```

Reading

- Efron and Hastie: Chapter 8, 17 and 18.

Quotation

"The world is yours, as well as ours, but in the last analysis, it is yours. You young people, full of vigor and vitality, are in the bloom of life, like the sun at eight or nine in the morning. Our hope is placed on you."

—Mao Zedong, Talk at a meeting with Chinese students and trainees in Moscow (November 17, 1957).

References

Athey, Susan, Julie Tibshirani, and Stefan Wager. 2019. "Generalized Random Forests." *The Annals of Statistics* 47 (2). Institute of Mathematical Statistics: 1148–78.

Breiman, Leo. 1996. "Bagging Predictors." *Machine Learning* 24 (2). Springer: 123–40.

———. 2001. "Random Forests." *Machine Learning* 45 (1). Springer: 5–32.

Breiman, Leo, Jerome H Friedman, Richard A Olshen, and Charles J Stone. 1984. *Classification and Regression Trees*. Wadsworth Publishing.

Gu, Shihao, Bryan Kelly, and Dacheng Xiu. 2018. "Empirical Asset Pricing via Machine Learning." National Bureau of Economic Research.

Hirano, Keisuke, and Jonathan H Wright. 2017. "Forecasting with Model Uncertainty: Representations and Risk Reduction." *Econometrica* 85 (2). Wiley Online Library: 617–43.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2 (5). Elsevier: 359–66.

Inoue, Atsushi, and Lutz Kilian. 2008. "How Useful Is Bagging in Forecasting Economic Time Series? A Case Study of Us Consumer Price Inflation." *Journal of the American Statistical Association* 103 (482). Taylor & Francis: 511–22.

Scornet, Erwan, Gérard Biau, and Jean-Philippe Vert. 2015. "Consistency of Random Forests." *The Annals of Statistics* 43 (4). Institute of Mathematical Statistics: 1716–41.

Shi, Zhentao. 2016. "Econometric Estimation with High-Dimensional Moment Equalities." *Journal of Econometrics* 195 (1). Elsevier: 104–19.

Wager, Stefan, and Susan Athey. 2018. "Estimation and Inference of Heterogeneous Treatment Effects Using Random Forests." *Journal of the American Statistical Association* 113 (523). Taylor & Francis: 1228–42.