Nonlinear example

Contents

Deep Feature Selection

In this markdown, we will demonstrate the comparison methods that are implemented in Table 2 of our paper.

User Guide on nonlinear example

In this example, a high dimensional dataset with 500 covariates and 300 observations is generated using the following equation.

$$y = \begin{cases} 1, & e^{x_1} + x_2^2 + 5\sin(x_3 x_4) - 3 > 0 \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

i.e. among 500 covariates, only the first 4 variables actually contributed to the response. Our task is to correctly select the important variables. Please see section 5.2 of the paper for detailed generation process.

In the markdown, the following methods will be implemented:

- Generalized Additive Model(GAM)
- Random Forest(RF)
- Bayesian Additive Regression Trees(BART)
- Bayesian Neural Networks(BNN)

SCAD here is only used for benchmark as the best method in linear example, and tell it has totally failed in this example. Thus the details will not be demonstrated here. Please find linear example in `../docs/markdowns/`.

Data Preparation

In this section, we will read in the data that is generated using `nonlinear_generator` from `./src/utils.py`

```
source("../../src/utils.R")
```

```
dirc = "../../data/nonlinear/p_500_N_600_s_4/"
k = 0 # dataset index from 0 to 9
X <- read.table(paste(dirc, 'X_', toString(k), '.txt', sep=""))
y <- read.table(paste(dirc, 'y_', toString(k), '.txt', sep=""))
train_pos_idx = which(y == 1)[1:150] # take 150 positive observations for training set
train_neg_idx = which(y == 0)[1:150] # take 150 negative observations for training set
test_pos_idx = which(y == 1)[151:300] # take rest positive observations for test set
test_neg_idx = which(y == 0)[151:300] # take rest negative observations for test set
train_idx = sort(cbind(train_pos_idx, train_neg_idx)) # bind training sample index
test_idx = sort(cbind(test_pos_idx, test_neg_idx)) # bind test sample index
X_train = X[train_idx,] # extract training set
```

```
y_train = y[train_idx,] # extract test set
X_test = X[test_idx,]
y_test = y[test_idx,]
N = dim(X_train)[1]
p = dim(X_train)[2]
```

The dimension of training set: (300, 500)

- The number of positive samples: 300
- The number of negative samples: 300

The dimension of test set: (300, 500)

- The number of positive samples: 300
- The number of negative samples: 300

Generalized Additive Model (GAM)

In this section, we will implement Generalized Additiveve Model(GAM) for variable selections and predictive preformance. We will use R package ***gamsel***. We first use function `gamsel` fit regularization path. We then calculate BICs with respect to the regaluraization path and select the best λ, and adopt the corresponding fitted model. `getActive` will help get the estimated support.

```
library(gamsel)
gam = gamsel(X_train, y_train, family="binomial")

SUPPs = getActive(gam, c(1:50))
Ss = as.numeric(lapply(SUPPs, length))
Y_Fits = predict(gam, X_train, type="response")
LOSSes = apply(Y_Fits, 2, cross_entropy, y_true=y_train)
BICs = BIC(LOSSes, Ss, N)
best_idx = which.min(BICs)

supp_gam = getActive(gam, index=c(best_idx), type="nonlinear")[[1]]
train_err_gam = 1 - (sum((Y_Fits[, best_idx]>=0.5)*1==y_train)/300.)
Y_Preds = predict(gam, X_test, type="response")
test_err_gam = 1 - (sum((Y_Preds[, best_idx]>=0.5)*1==y_test)/300.)
```

The selected support is 1, 2, 3, 4, 480, the training error is 0.1266667, the test error is 0.14.

Random Forest

In this section, we will implement Random Forest(RF) for predictive performance and variable importance. We will use R package ***h2o***. For the installation, please see ***h2o***. To use `h2o`, you need to initial a session by calling `h2o.init()`, and make sure all your data is in `h2o` objects. Then function `h2o.randomForest` can be used for the training. Here we take the default settings.

```
library(h2o)
h2o.init() # start a h2o session
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /scratch/tmp/RtmpxRnXLG/file5dadbac2af4/h2o_chenya3i_started_from_r.out
```

```
##     /scratch/tmp/RtmpxRnXLG/file5dad94d8cb4/h2o_chenya3i_started_from_r.err
##
##
## Starting H2O JVM and connecting: ..... Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         4 seconds 932 milliseconds
##     H2O cluster timezone:       Europe/Prague
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.0.6
##     H2O cluster version age:    26 days
##     H2O cluster name:           H2O_started_from_R_chenya3i_qws292
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   28.98 GB
##     H2O cluster total cores:    1
##     H2O cluster allowed cores:  1
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 3.6.1 (2019-07-05)
```

```
h2o.no_progress()
# Transfer data to h2o objects
data_train = as.h2o(cbind(X_train, y_train))
data_train["y_train"] = as.factor(data_train["y_train"])
data_test = as.h2o(cbind(X_test, y_test))
data_test["y_test"] = as.factor(data_test["y_test"])

rf = h2o.randomForest(y="y_train", training_frame=data_train, seed=1)
supp_rf = c(1:500)[h2o.varimp(rf)$percentage>0.014]
fit_rf = predict(rf, as.h2o(X_train))$predict
pred_rf = predict(rf, as.h2o(X_test))$predict
train_err_rf = 1 - sum(fit_rf==as.h2o(y_train))/300.
test_err_rf = 1 - sum(pred_rf==as.h2o(y_test))/300.
```

The selected support is 1, 2, 3, 4, the training error is 0.01, the test error is 0.27.

Bayesian Additive Regression Trees(BART)

In this section, we will implement Bayesian Additive Regression Trees (BART) for predictive performance and variable selection. We will use R package ***bartMachine***. However, this methods take significant time and resource. You can set up some options, like memory size and number of cores used, to speed up the computing. Function `bartMachine` is used for the training and function `var_selection_by_permute_cv` is used for selecting variables.

```
options(java.parameters = "-Xmx5g") # reserve 5G memory
library(bartMachine)
set_bart_machine_num_cores(4)
```

```
## bartMachine now using 4 cores.
```

```
bart_vs = bartMachine(X=X_train, y=(y_train), num_trees=75, seed=1)
```

```
## bartMachine initializing with 75 trees...
## bartMachine vars checked...
## bartMachine java init...
## bartMachine factors created...
## bartMachine before preprocess...
## bartMachine after preprocess... 501 total features...
## warning: cannot use MSE of linear model for s_sq_y if p > n. bartMachine will use sample var(y) inste
## bartMachine sigsq estimated...
## bartMachine training data finalized...
## Now building bartMachine for regression ...
## evaluating in sample data...done
```

```
var_sel = var_selection_by_permute_cv(bart_vs)
```

```
## cv #1
## avg.....null.....................................................................................
## method...
## cv #2
## avg.....null.....................................................................................
## method...
## cv #3
## avg.....null.....................................................................................
## method...
## cv #4
## avg.....null.....................................................................................
## method...
## cv #5
## avg.....null.....................................................................................
## method...
## final
## avg.....null.....................................................................................
```

```
bart = bartMachine(X=X_train, y=factor(y_train), num_trees=75, seed=1)
```

```
## bartMachine initializing with 75 trees...
## bartMachine vars checked...
## bartMachine java init...
## bartMachine factors created...
## bartMachine before preprocess...
## bartMachine after preprocess... 501 total features...
## warning: cannot use MSE of linear model for s_sq_y if p > n. bartMachine will use sample var(y) inste
## bartMachine sigsq estimated...
## bartMachine training data finalized...
## Now building bartMachine for classification ...
## evaluating in sample data...done
```

```
supp_bart = var_sel$important_vars_cv
fit_bart = predict(bart, X_train, type="class")
pred_bart = predict(bart, X_test, type="class")
train_err_bart = 1 - sum(fit_bart==y_train)/300.
test_err_bart = 1 - sum(pred_bart==y_test)/300.
```

The selected support is V1, V3, V4, the training error is 0.0566667, the test error is 0.2166667.

Bayesian Neural Networks

In this section, we will implement Bayesian Neural Network (BNN) for predictive performance and variable selection. We will use R package ***BNN***. This method take significant time to run in markdown. We only demonstrate the code here and load pre-trained model in `../../outputs/models/` to show the results. `BNNsel` function is not reproducable as there is no arugment of `seed` in it. We tried to set seed at the begining of the script, but was unable to get the exact same model.

```
library(BNN)
X_data = rbind(X_train, X_test)
y_data = as.factor(c(y_train, y_test))
bnn = BNNsel(X_data, y_data, train_num=300, hid_num=3, lambda=0.5,
             total_iteration=1000000, popN=20, nCPUs=10)
```

```
load('../../outputs/models/bnn_0_0.2.RData')
supp_bnn = c(1:500)[bnn$mar[2:501] == 1] # bnn includes intercept
train_err_bnn = 1 - sum((bnn$fit>0.5)*1 == y_train)/300.
test_err_bnn = 1 - sum((bnn$pred>0.5)*1 == y_test)/300.
```

The selected support is 1, 2, 3, 4, 194, 477, the training error is 0.02, the test error is 0.1566667.