

baseline_main

Yufei Zhao

2017/10/29

```
print(R.version)
```

```
##  
## platform      _  
## arch          x86_64-mingw32  
## os            mingw32  
## system        x86_64, mingw32  
## status  
## major         3  
## minor         4.2  
## year          2017  
## month         09  
## day           28  
## svn rev       73368  
## language      R  
## version.string R version 3.4.2 (2017-09-28)  
## nickname      Short Summer
```

```
if(!require("gbm")){  
  install.packages("gbm")  
}
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
## Loading required package: lattice
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```
if(!require("dplyr")){  
  install.packages("dplyr")  
}
```

```
## Loading required package: dplyr
```

```
##  
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':  
##  
## filter, lag
```

```
## The following objects are masked from 'package:base':  
##  
## intersect, setdiff, setequal, union
```

```
library(gbm)  
library(dplyr)
```

Step 0: set seed.

In order to obtain reproducible results, `set.seed()` whenever randomization is used.

```
set.seed(1)
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (T/F) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

```
run.cv=TRUE # run cross-validation on the training set  
K <- 3 # number of CV folds  
#run.feature.train=TRUE # process features for training set  
run.test=TRUE # run evaluation on an independent test set  
#run.feature.test=TRUE # process features for test set
```

Using cross-validation or independent test set evaluation, we compare the performance of different classifiers or classifiers with different specifications. In this example, we use GBM with different `depth`. In the following chunk, we list, in a vector, setups (in this case, `depth`) corresponding to models that we will compare. In your project, you maybe comparing very different classifiers. You can assign them numerical IDs and labels specific to your project.

```
model_values <- seq(3, 11, 4)
model_labels = paste("GBM with depth =", model_values)
```

Read in data

```
# lables(0 for muffin, 1 for chicken, 2 for dog)
labels <- read.csv("../data/label_train.csv",header=TRUE)
colnames(labels)[2] <- "labels"
```

```
sift_data <- read.csv("../data/sift_train.csv",header=TRUE, stringsAsFactors = FALSE)
```

divide into training and sampling data set

```
sampling <- sample(seq(1:3000),3000*0.25) %>% sort()
label_test <- labels[sampling,-1]
label_train <- labels[-sampling,-1]
dat_test <- sift_data[sampling,-1]
dat_train <- sift_data[-sampling,-1]
```

load train and test method

```
source("../lib/baseline_train.R")
source("../lib/baseline_test.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in this example.

```
# small range case !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#dat_train <- sift_data[1:100,-1]
#label_train <- labels[1:100,-1]
#dat_test <- sift_data[101:200,-1]
#label_test <- labels[101:200,-1]
```

```

source("../lib/baseline_cv.R")

if(run.cv){
  err_cv <- array(dim=c(length(model_values), 2))
  for(k in 1:length(model_values)){
    cat("k=", k, "\n")
    err_cv[k,] <- cv.function(dat_train, label_train, model_values[k], K)
  }
  save(err_cv, file="../output/baseline_err_cv.RData")
}

```

```

## k= 1
## k= 2
## k= 3

```

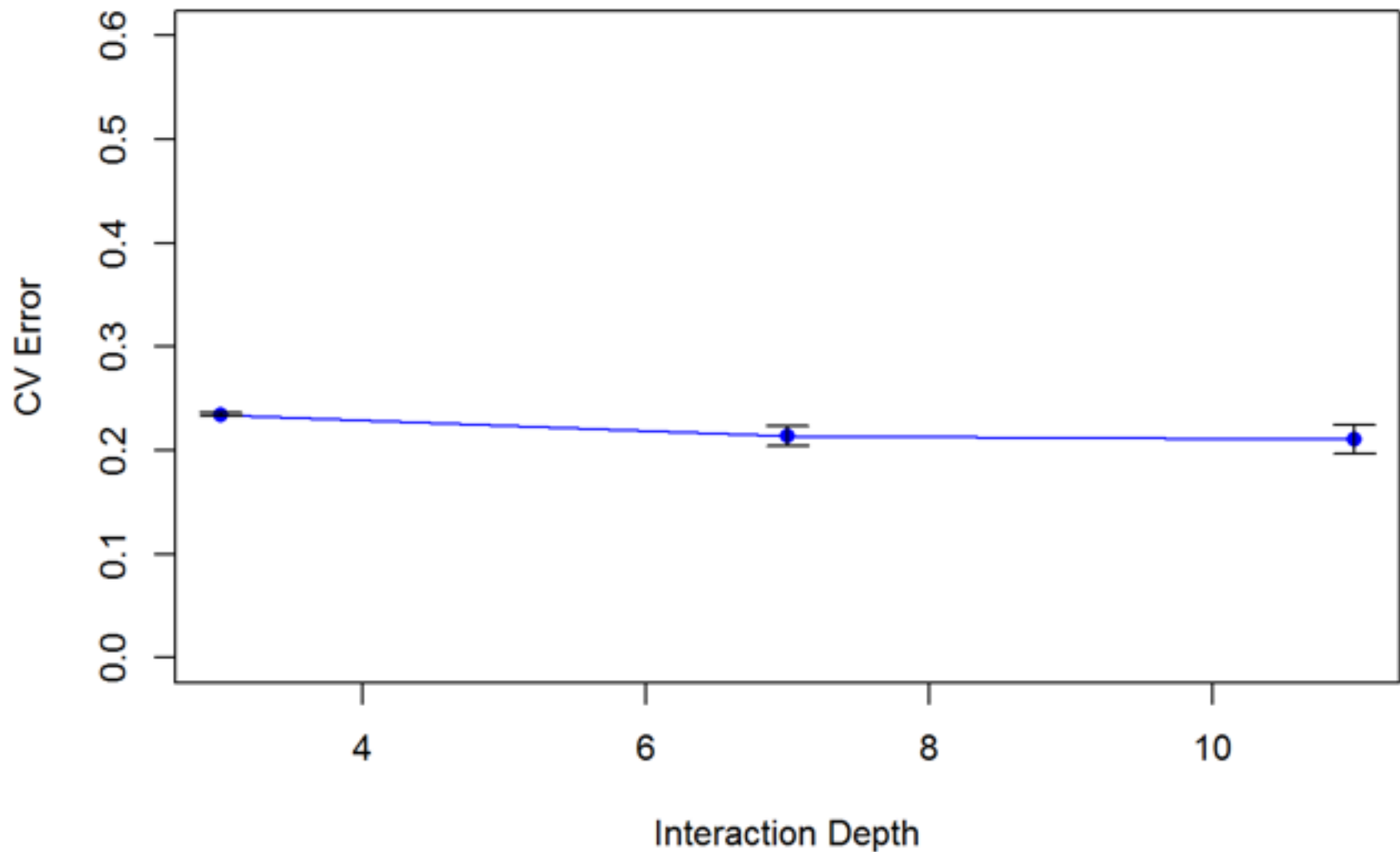
Visualize cross-validation results.

```

if(run.cv){
  load("../output/baseline_err_cv.RData")
  #pdf("../fig/cv_results.pdf", width=7, height=5)
  plot(model_values, err_cv[,1], xlab="Interaction Depth", ylab="CV Error",
       main="Cross Validation Error", type="n", ylim=c(0, 0.6))
  points(model_values, err_cv[,1], col="blue", pch=16)
  lines(model_values, err_cv[,1], col="blue")
  arrows(model_values, err_cv[,1]-err_cv[,2], model_values, err_cv[,1]+err_cv[,2],
        length=0.1, angle=90, code=3)
  #dev.off()
}

```

Cross Validation Error



- Choose the “best” parameter value

```
model_best=model_values[1]
if(run.cv){
  model_best <- model_values[which.min(err_cv[,1])]
}

par_best <- list(depth=model_best)
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

```
tm_train=NA
tm_train <- system.time(fit_train <- train(dat_train, label_train, par_best))
```

```
## Warning in gbm.fit(x = dat_train, y = label_train, n.trees = 1000,
## distribution = "multinomial", : variable 1012: V1012 has no variation.
```

```
## Warning in gbm.fit(x = dat_train, y = label_train, n.trees = 1000,
## distribution = "multinomial", : variable 2688: V2688 has no variation.
```

```
## Warning in gbm.perf(fit_gbm, method = "OOB", plot.it = FALSE): OOB
## generally underestimates the optimal number of iterations although
## predictive performance is reasonably competitive. Using cv.folds>0 when
## calling gbm usually results in improved predictive performance.
```

```
save(fit_train, file="../output/baseline_fit_train.RData")
```

Step 5: Make prediction

Feed the final training model with the completely holdout testing data.

```
tm_test=NA
if(run.test){
  load(file="../output/baseline_fit_train.RData")
  tm_test <- system.time(pred_test <- test(fit_train, dat_test))
  save(pred_test, file="../output/baseline_pred_test.RData")
}
```

Calculate the prediction rate

```
load(file="../output/baseline_pred_test.RData")
correct_rate <- mean(pred_test == label_test)
cat("Correct rate for prediction = ", correct_rate[1], "\n")
```

```
## Correct rate for prediction = 0.7546667
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```
cat("Time for training model=", tm_train[1], "s \n")
```

```
## Time for training model= 2775.39 s
```

```
cat("Time for making prediction=", tm_test[1], "s \n")
```

```
## Time for making prediction= 0.11 s
```