EnsembleModels

Lisa

March 12, 2015

Introduction

I applied a package **caretEnsemble** for making ensembles of caret models. It written by *Zach Mayer* and his gitbut website is https://github.com/zachmayer/caretEnsemble. The **caretEnsemble** includes 2 different algorithms for combining models:

1. Greedy stepwise ensembles (returns a weight for each model), using **caretEnsemble**.
2. Stacks of caret models, using **caretStack**. The stacking algorithm simply builds a second caret model on top of the existing models (using their predictions as input).

Resourse:

- Binary classification model with caretEnsemble on gist of *Zach Mayer* https://gist.github.com/zachmayer/5179418/
- A brief introduction to caretEnsemble http://cran.r-project.org/web/packages/caretEnsemble/vignettes/caretEnsemble-intro.html

Load libraries

```
##          used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 293108 15.7     467875 25.0   293108 15.7
## Vcells 522304  4.0    1031040  7.9   522304  4.0
```

I deleted missing values (NA) in the dataset and added levels for response as "good" and "bad".

```
load("data_17_Feb.RData")
data=data[,2:56]

# delete 2557 missing values in "compt_size"
sum(is.na(data$compt_size))
```

```
## [1] 2557
```

```
data=na.omit(data)

dim(data)
```

```
## [1] 13717    55
```

```
sum(is.na(data)) #check missing value
```

```
## [1] 0
```

```r
#make levels for response
levels(data$class)=c("good","bad")
str(data)
```

```
## 'data.frame':    13717 obs. of  55 variables:
##  $ class                : Factor w/ 2 levels "good","bad": 1 2 1 1 1 1 2 1 2 2 ...
##  $ doc_id               : num  1 1 1 1 1 1 2 0 0 2 ...
##  $ item_price           : int  3900 4750 4580 2699 4900 8200 5399 5488 2998 3900 ...
##  $ rate                 : num  0.0401 0.0386 0.0389 0.0389 0.0414 ...
##  $ first_pay            : int  0 1200 1000 0 1300 2500 1800 1500 600 0 ...
##  $ loan_tot             : int  3900 3550 3580 2699 3600 5700 3599 3988 2398 3900 ...
##  $ m_pay                : int  373 433 537 285 299 545 439 436 360 412 ...
##  $ loan_time            : int  18 12 9 15 24 18 12 15 9 15 ...
##  $ live_t               : num  5.48 4.8 2.56 1.95 1.95 ...
##  $ tel_time             : num  1.61 4.41 3.22 0 4.26 ...
##  $ income               : num  8.29 8.01 8.29 7.82 8.16 ...
##  $ work_time            : num  1.95 2.56 2.56 1.95 2.08 ...
##  $ compt_size           : int  50 5 5 1000 1000 20 1000 50 20 10 ...
##  $ loan_date            : int  20141219 20141112 20141231 20150112 20141227 20141219 20141217 2015011
##  $ age                  : int  27 32 32 20 26 24 31 25 34 24 ...
##  $ bank_name            : Factor w/ 14 levels "AAA","AAB","AAC",..: 2 1 1 1 3 2 2 5 2 5 ...
##  $ ins                  : Factor w/ 2 levels "AAA","AAB": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sex                  : Factor w/ 2 levels "AAA","AAB": 1 2 1 1 1 1 1 2 1 1 ...
##  $ reg_type             : Factor w/ 3 levels "AAA","AAB","MISS": 1 1 1 1 3 1 1 1 1 1 ...
##  $ reg_vs_live          : Factor w/ 2 levels "AAA","AAB": 2 2 1 1 1 1 1 1 1 2 ...
##  $ live_type            : Factor w/ 7 levels "AAA","AAB","AAC",..: 5 5 1 1 1 1 1 4 3 4 ...
##  $ tel_fee              : Factor w/ 4 levels "AAA","AAB","AAC",..: 3 1 1 4 2 1 2 1 1 2 ...
##  $ real_name            : Factor w/ 3 levels "AAA","AAB","MISS": 2 2 2 3 1 1 2 1 1 1 ...
##  $ marrage              : Factor w/ 6 levels "AAA","AAB","AAC",..: 2 3 4 2 2 2 3 2 4 2 ...
##  $ live_vs_reg          : Factor w/ 3 levels "AAA","AAB","MISS": 1 1 2 1 1 1 1 2 1 2 ...
##  $ real_home_tel        : Factor w/ 2 levels "AAA","MISS": 2 2 2 2 2 2 2 2 1 2 ...
##  $ contact_realtion     : Factor w/ 18 levels "AAA","AAB","AAC",..: 3 1 3 2 2 2 3 3 1 2 ...
##  $ edu                  : Factor w/ 10 levels "AAA","AAB","AAC",..: 5 1 1 5 4 5 4 4 1 2 ...
##  $ yn_loan              : Factor w/ 4 levels "AAA","AAB","AAC",..: 1 1 1 1 2 1 1 1 1 1 ...
##  $ compt_type           : Factor w/ 13 levels "AAA","AAB","AAC",..: 1 1 1 1 1 1 3 13 1 1 ...
##  $ compt_industry       : Factor w/ 27 levels "AAA","AAB","AAC",..: 3 2 1 8 3 5 4 20 8 1 ...
##  $ shop_level           : Factor w/ 3 levels "AAA","AAB","AAC": 2 1 1 3 1 1 3 3 1 1 ...
##  $ custom_type          : Factor w/ 2 levels "AAA","AAB": 1 1 1 1 1 1 1 1 1 1 ...
##  $ history              : Factor w/ 21 levels "AAA","AAB","AAC",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ id_info              : Factor w/ 4 levels "AAA","AAB","AAC",..: 1 4 1 1 1 1 1 1 1 4 ...
##  $ item_info            : Factor w/ 7 levels "AAA","AAB","AAC",..: 1 7 1 1 1 1 1 1 1 7 ...
##  $ tel_num_chk_info     : Factor w/ 12 levels "AAA","AAB","AAC",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ compt_tel_chk_info   : Factor w/ 10 levels "AAA","AAB","AAC",..: 10 3 1 1 2 10 2 10 10 2 ...
##  $ loan_info            : Factor w/ 4 levels "AAA","AAB","AAC",..: 1 4 1 1 1 1 1 1 1 4 ...
##  $ compt_name_chk_info  : Factor w/ 7 levels "AAA","AAB","AAC",..: 1 1 2 2 2 2 1 1 1 1 ...
##  $ loan_num_level       : Factor w/ 5 levels "AAA","AAB","AAC",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ item_price_chk       : Factor w/ 13 levels "AAA","AAB","AAC",..: 1 1 1 1 1 1 1 3 1 1 ...
##  $ tel_chek             : Factor w/ 24 levels "AAA","AAB","AAC",..: 3 8 3 3 9 3 9 3 8 6 ...
##  $ compt_contact_tel_chk: Factor w/ 10 levels "AAA","AAB","AAC",..: 2 3 2 2 2 2 2 2 2 2 ...
##  $ QQ_chk               : Factor w/ 19 levels "AAA","AAB","AAC",..: 3 7 1 7 7 7 2 7 1 1 ...
##  $ compt_tel_chk        : Factor w/ 18 levels "AAA","AAB","AAC",..: 18 1 2 2 5 18 6 18 18 6 ...
##  $ live_info            : Factor w/ 4 levels "AAA","AAB","AAC",..: 1 4 1 1 1 1 1 1 1 4 ...
##  $ compt_contact_chk    : Factor w/ 18 levels "AAA","AAB","AAC",..: 1 3 1 6 1 6 4 6 1 4 ...
```

```
##  $ cus_tel_chk          : Factor w/ 11 levels "AAA","AAB","AAC",..: 4 6 5 4 2 5 4 2 6 5 ...
##  $ ass_chk              : Factor w/ 5 levels "AAA","AAB","AAC",..: 5 1 1 1 1 2 1 2 1 1 ...
##  $ live_info_chk        : Factor w/ 4 levels "AAA","AAB","AAC",..: 1 4 1 1 1 1 1 1 1 4 ...
##  $ home_contact_info    : Factor w/ 5 levels "AAA","AAB","AAC",..: 1 5 5 1 2 1 2 2 2 5 ...
##  $ reg_place            : Factor w/ 7 levels "AAA","AAC","AAD",..: 4 1 1 2 1 1 1 4 4 6 ...
##  $ live_place           : Factor w/ 7 levels "AAA","AAB","AAC",..: 1 5 5 3 5 5 5 1 5 6 ...
##  $ work_place           : Factor w/ 7 levels "AAA","AAB","AAC",..: 1 5 5 3 5 5 5 1 5 6 ...
##  - attr(*, "na.action")=Class 'omit'  Named int [1:2557] 1 2 22 26 38 42 49 53 54 56 ...
##   .. ..- attr(*, "names")= chr [1:2557] "1360" "10517" "9683" "1792" ...
```

```r
prop.table(table(data$class)) #proportions of our outcome variable
```

```
##
##      good       bad
## 0.6380404 0.3619596
```

Our data isn't perfectly balanced but it certainly isn't skewed or considered a rare event.

I added levels for response, because when you ask for class probabilities, model predictions are a data frame with separate columns for each class/level. If *class* doesn't have levels, `data.frame` converts them to valid names, which creates a problem because a different (but valid) name of levels.

Data splitting for training and testing

```
## [1] 10288    55
```

```
## [1] 3429   55
```

Model tuning

```r
#Setup CV Folds
#returnData=FALSE saves some space
folds=10
repeats=3
# Try using ROC (AUC) as metric since that is what caretEnsemble uses
#trainMetric = "ROC" # Classification, required for some two class analyses

myControl <- trainControl(method='cv', number=folds, repeats=repeats,
                          returnResamp='none', classProbs=TRUE,
                          returnData=FALSE, savePredictions=TRUE,
                          verboseIter=TRUE, allowParallel=TRUE,
                          summaryFunction=twoClassSummary,
                          # For caretEnsemble, make the resampling indexes all the same.
                          index=createMultiFolds(train$class, k=folds, times=repeats),

                          )
PP <- c('center', 'scale')
```

```r
set.seed(800)
detectCores()
registerDoParallel(48,cores=48)
getDoParWorkers()

model_list<- caretList(
  class~., data=train,
  trControl=myControl,
  methodList=c('blackboost', 'parRF'),
  tuneList=list(
    gbm=caretModelSpec(method='gbm',
                       tuneGrid=expand.grid(.n.trees=c(1:10)*10, .interaction.depth=1, .shrinkage = 0.1
                       ),
    mlpweightdecay=caretModelSpec(method='mlpWeightDecay', trace=FALSE, preProcess=PP),
    earth=caretModelSpec(method='earth', preProcess=PP),
    glm=caretModelSpec(method='glm', preProcess=PP),
    svmRadial=caretModelSpec(method='svmRadial', preProcess=PP),
    knn=caretModelSpec(method='knn',preProcess=PP),
    gam=caretModelSpec(method='gam', preProcess=PP),
    glmnet=caretModelSpec(method='glmnet', preProcess=PP)
    )
  )
```

Benchmark models (candidate)

We considered 9 candidate models. Some are the least interpretable and most flexible, such as *boosted trees*, *Stochastic Gradient Boosting* or *support vector machines* which have a high likelihood of producing the most accurate results. Others are simpler models that are less opaque (e.g., not complete black boxes), such as *multivariate adaptive regression splines (MARS)* or *generalized additive models*. The model and argument value are in `names(model_list)` and `names(infor)`, respectively. The more detail method information such as tuning parameters can be explored on caret model list.

```r
load("modellist3") #"model_list" is saved in "modellist3"

#Make a list of all the models
names(model_list) = sapply(model_list, function(x) x$method) #method names
infor = sapply(model_list, function(x) x$modelInfo$label) #label of methods
names(infor) = infor
names(model_list)
```

```
## [1] "gbm"            "mlpWeightDecay" "earth"          "glm"
## [5] "svmRadial"      "knn"            "gam"            "glmnet"
## [9] "blackboost"
```

```r
names(infor)
```

```
## [1] "Stochastic Gradient Boosting"
## [2] "Multi-Layer Perceptron"
## [3] "Multivariate Adaptive Regression Spline"
## [4] "Generalized Linear Model"
## [5] "Support Vector Machines with Radial Basis Function Kernel"
```

```
## [6] "k-Nearest Neighbors"
## [7] "Generalized Additive Model using Splines"
## [8] "glmnet"
## [9] "Boosted Tree"
```

```r
sort(sapply(model_list, function(x) max(x$results$ROC))) #maximum accuracy of each method
```

```
##           knn mlpWeightDecay            gbm            gam      svmRadial
##     0.7385780      0.8439986      0.8484180      0.8551207      0.8562848
##        glmnet     blackboost            glm          earth
##     0.8586937      0.8589403      0.8618615      0.8646148
```

Make a greedy ensemble

```r
#Make a greedy ensemble - currently can only use RMSE
greedy <- caretEnsemble(model_list, iter=1000L)
sort(greedy$weights, decreasing=TRUE)
```

```
## blackboost        glm      earth  svmRadial        knn
##      0.403      0.235      0.233      0.112      0.017
```

```r
#summary(greedy)
greedy$error
```

```
##                    [,1]
## good vs. bad 0.8730575
## attr(,"names")
## [1] "AUC"
```

The greedy model relies 87% on the **blackboost**, **glm** and **earth**, which makes sense as these are top three highest accuracte models on the training set. The ensemble's AUC (area under the curve) on the training set is 0.87, which is about 1% better than the best individual model (**blackboost**).

Make a linear regression ensemble

```r
#Make a linear regression ensemble
linear <- caretStack(model_list, method='glm', trControl=trainControl(method='cv'))
summary(linear$ens_model$finalModel)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5860  -0.5928  -0.3628   0.5488   2.5868
```

```
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -3.85030    0.18240 -21.109  < 2e-16 ***
## gbm            -1.91505    0.30657  -6.247 4.19e-10 ***
## mlpWeightDecay -0.17920    0.12451  -1.439 0.150091
## earth           1.15508    0.16601   6.958 3.45e-12 ***
## glm             1.84366    0.17454  10.563  < 2e-16 ***
## svmRadial       0.67536    0.18285   3.693 0.000221 ***
## knn             0.31243    0.09782   3.194 0.001403 **
## gam            -0.55003    0.13966  -3.938 8.21e-05 ***
## glmnet         -0.33115    0.29796  -1.111 0.266407
## blackboost      6.89579    0.50825  13.568  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 40404  on 30863  degrees of freedom
## Residual deviance: 26043  on 30854  degrees of freedom
## AIC: 26063
##
## Number of Fisher Scoring iterations: 5
```

```
linear$error
```

```
##   parameter  Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.8073477 0.5720676 0.00789496 0.01809372
```

```
CF <- coef(linear$ens_model$finalModel)[-1] #coefficient of glm model
CF/sum(CF) #glm model weights
```

```
##            gbm mlpWeightDecay          earth            glm      svmRadial
##    -0.24220050    -0.02266333     0.14608536     0.23317168     0.08541362
##            knn            gam         glmnet     blackboost
##     0.03951412    -0.06956343    -0.04188097     0.87212346
```

The linear model uses all of the models, and achieves an AUC of 0.81, which lower than the average individual model accuracy. I'm not sure if this is a failure of the stacking model, because accuracy becomes much higher later on in the test set prediction.

Different from greedy model, The glm-weighted model weights relies most on only **blackboost**, and the weight is 0.87.

Result (predict for test set)

```
library(caTools)
model_preds <- lapply(model_list, predict, newdata=test, type='prob')
model_pred <- lapply(model_preds, function(x) x[,'good'])
model_pred <- data.frame(model_pred)
```

```r
ENS_greedy <- predict(greedy, newdata=test)
model_pred$ENS_greedy=1-ENS_greedy
ENS_linear= predict(linear, newdata=test,type='prob')
model_pred$ENS_linear <- ENS_linear[,"good"]
```

```r
library(caTools)
load("pred")  #"model_pred" is saved in "pred"
sort(data.frame(colAUC(model_pred, test$class)))
```

```
##                    knn mlpWeightDecay       gbm svmRadial       gam
## good vs. bad 0.7465118      0.8368436 0.8476753 0.8503573 0.8524289
##              blackboost       glm    glmnet     earth ENS_linear
## good vs. bad  0.8543263 0.8544994 0.8557147 0.8576826  0.8673406
##              ENS_greedy
## good vs. bad  0.8678058
```

The greedy model testing prediction still better than any individual model prediction, and has the highest accuracy 0.87.

Most predictions in test set are reasonable, except "linear regression ensemble", I mentioned before, that it has much higher testing accuracy 0.867 than training accuracy 0.81. *I've no idea why it happened.*