Data Mining Final

응용통계학과 20152410 배형준

Data Mining I: Final Examination (Spring 2020)
Friday, June 19, 12:00 AM - Tuesday, June 23, 23:59 PM (100%)
Wednesday, June 24, 12:00 AM - 23:59 PM (70%)
Thursday, June 25, 12:00 AM - 23:59 PM (40%)

- **Upload a single pdf file including all your work for all problems once.** If you upload multiple files, then only the last uploaded file will be counted. For example, you upload a file on June 23 and upload another file on June 24, then the file submitted on June 23 will be ignored.
- **You are fully responsible for uploading your work.** Try to upload your file as early as possible. There may be heavy internet traffic around the deadline. The above deadline is the time you finish uploading. It is not the time you start uploading.
- **Do not email your exam.**
- **Your pdf file must include this page with your name and CAU ID number.**
- **Show all work to obtain full credits. Grading is based on what you write, not on what you think.**

(10 points) This is to certify that I have used no other person as a resource. In addition, I have read the above instructions. I fully understand the instructions and agree on the terms.

Name: **Bae Hyungjun** ID# : **20152410**

Please choose Accuracy or AUC to calculate your exam score.

I want to use **AUC.**

Use the attached “Train.csv” data to classify the binary response variable y. Follow the writing instructions posted in eClass. There are two more files that are posted in eClass. “Xtest.csv” file has the predictor values that you need to calculate their predicted classes. The other file “Ans.csv” has two columns: one is “yhat” and the second is “prob”. You fill the predicted classes and the posterior probabilities from your final model. Upload (1) your pdf file including all your work following the writing instructions and (2) your “Ans.csv” file that you fill two columns in the same ID order as “Xtest.csv” ID column. In addition, choose Accuracy or AUC to determine a part of your exam scores. Note that you are responsible for using the consistent coding and order as the original data when you fill the “Ans.csv” file.

grading items	points	policy
signed 1st page	10	Included as 1st page with name, ID, and your choice of Accuracy or AUC?
writing	10	Based on the writing instructions
methods and program	10	How many methods? Are they properly applied to the problem? Does the attached program run without error? Does the program produce the same results in your report? etc.
Accuracy or AUC	30	[(Accuracy or AUC) - 85] x 2 based on your “Ans.csv” file. If Accuracy or AUC is less than 85, then the score will be zero.

Contents

Main text

Sequence of data analysis

1. load dataset, split dataset as train and validation using stratifed partition function
2. data modeling using caret to tune hyper parameter of candidate models
3. evaluate the models
4. choose models to use or to make ensemble
5. confime final model, predict class and posterior probability of Xtest

1. load and split dataset

read.csv 함수를 이용하여 train, test 를 불러온 뒤 train 을 다시 train : validation = 0.75 : 0.25 비율로 train 과 validation 으로 나누었습니다. 나눌 때 sample 을 이용하여 4000 개의 데이터를 무작위로 나누지 않고, createDataPartition 함수를 이용하여 response variable 의 0 과 1 의 비율을 유지하여 나눠주었습니다. 이는 train 과 validation 을 비슷한 상태로 만들어 validation 을 이용해 train 을 학습한 모델을 더 잘 평가하기 위함입니다. 아래의 코드 결과는 4000 개의 trainset, 3001 개의 train, 999 개의 validation 의 response variable 의 0 과 1 의 비율입니다. 3 개의 데이터셋 모두 0 이 1 보다 약 1.5 배 많은 것을 확인할 수 있습니다.

```
##  ratio of trainset target : 1.533249
##  ratio of train target : 1.532489
##  ratio of validation target : 1.535533
```

2. data modeling

3001 개의 train 을 학습하기 위한 모델 후보로 glmnet (elastic net), support vector machine with radial kernel, random forest, xgboost 4 개의 모델을 선정하였습니다. 모델의 결과에 따라서 4 개 중 가장 좋은 모델을 선택할지 또는 복수의 모델을 앙상블할지 결정하려고 합니다. 모델의 학습으로 caret 패키지의 train 함수를 이용하였고, cross validation 의 fold 는 4 개, 튜닝 횟수는 250 번, 하이퍼 파라미터 탐색은 'random'으로 동일하게 설정한 뒤 학습하였습니다. 평가 기준을 AUC 로 선택했기 때문에 모델 평가 인수인 metric 을 'ROC'로 설정해주었습니다.

2-1. glmnet

학습을 위해 elastic net 를 위한 preProcess=c('center', 'scale')로 설정해주어 표준화된 데이터를 학습할 수 있도록 설정하였습니다. 학습 결과 alpha=0.9571266, lambda=0.001491034 를 최적 하이퍼 파라미터로 얻었습니다. alpha 가 1 에 가까운 것으로 보아 ridge 보다는 lasso 의 L1 penalized term 이 모델 학습에 큰 영향을 끼쳤다고 볼 수 있습니다. 최적 하이퍼 파라미터일 때의 CV AUC 는 0.9615987 입니다. 선형모델의 CV AUC 가 꽤 준수한 성능을 보여주기 때문에 이후에 사용할 복잡한 모델들이 더 좋은 성능을 보여줄 것이라 기대할 수 있었습니다.

```
##       alpha      lambda       ROC
## 1 0.9571266 0.001491034 0.9615987
```

2-2. support vector machine with rbf kernel

rbf 커널을 이용한 support vector machine 도 glmnet 과 마찬가지로 preProcess=c('center', 'scale')로 설정해주어 변수의 단위에 따른 영향력을 같게 만들어주었습니다. 학습 결과 sigma=0.009966466, C=21.71236 을 최적 하이퍼 파라미터로 얻었습니다. C 값이 큰 것으로 보아 모델이 오분류된 데이터에 대한 가중치를 높게 책정한 것을 확인할 수 있습니다. 최적 하이퍼 파라미터일때의 CV AUC 는 0.9678749 입니다. 이는 glmnet 의 CV AUC 보다 0.0062762 보다 크므로 더 좋은 분류기라고 평가할 수 있겠으나 그 차이가 미비하여 성능이 크게 차이 난다고 볼 수 없다고 생각합니다.

```
##        sigma        C       ROC
## 1 0.009966466 21.71236 0.9678749
```

2-3. random forest

method='rf'보다 빠르게 학습할 수 있는 method='ranger'를 이용하여 random forest classifier 를 학습하였습니다. 최적 하이퍼 파라미터로 min.node.size=4, mtry=4, splitrule='gini'를 얻었고 그때의 CV AUC 는 0.977648 입니다. 앞선 두 모델보다 성능이 살짝 더 향상된 것을 확인할 수 있습니다. splitrule 이 extratrees 가 아닌 것으로 보아 tree 의 절단점을 무작위로 자르는 것보단 어느 정도 절단점을 탐색한 후 최적 지점을 선정하는 것이 이 데이터에 대해선 좋은 방법이란 것을 확인할 수 있습니다.

```
##   min.node.size mtry splitrule      ROC
## 1             4    4      gini 0.977648
```

2-4. xgboost

method='xgbTree'를 이용하여 base learner 가 tree 인 xgboost classifier 를 학습하였습니다. 아래의 코드 결과와 같이 7 개의 최적 하이퍼 파라미터를 얻었습니다. 그때의 CV AUC 는 0.9800166 으로, 앞서 학습했던 glmnet, svmradial, random forest 보다 더 좋은 성능을 보여줍니다.

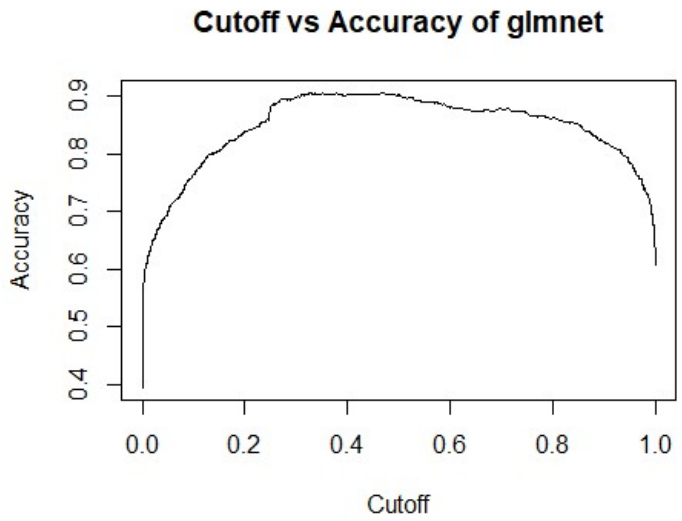
이제 4 개의 모델이 학습에 사용되지 않았던 validation 에 대해 어느 정도의 성능을 가지는지 확인해보도록 하겠습니다.

```
##         eta max_depth    gamma colsample_bytree min_child_weight subsample
## 1 0.09257654         9 1.453736        0.3281601                0 0.9524782
##   nrounds       ROC
## 1     338 0.9800166
```

3. evaluate the models

Validation 에 대한 AUC, maximum Accuracy 와 그때의 Cutoff 값을 이용하여 4 개의 모델을 평가하려 합니다. 또한 Cutoff vs Accuracy 그래프를 확인하여, 만약 앙상블을 할 경우 어떻게 cutoff 를 결정해야 하는지 생각해보도록 하겠습니다.

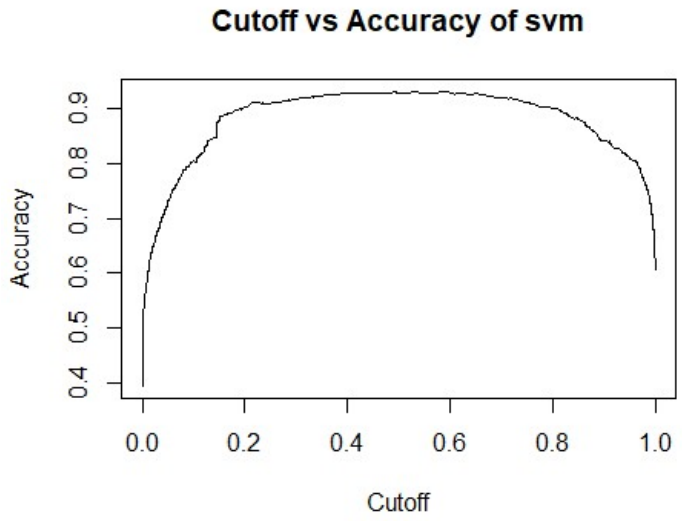
3-1. glmnet



Validation AUC 는 0.9628162, cutoff 가 0.326344 일 때 Accuracy 가 0.9059059 로 최댓값을 가집니다. 위의 그래프를 보면 Accuracy 가 cutoff 에 살짝 민감하게 반응하는 것처럼 보입니다. AUC 는 꽤 높은 편이라고 생각하지만 Accuracy 가 약 90.6%로 만족스럽지 않은 결과입니다.

```
##  AUC of glmnet : 0.9628162
##  Max Accuracy of glmnet : 0.9059059
##  Cutoff of maximum accuracy of glmnet : 0.326344
```

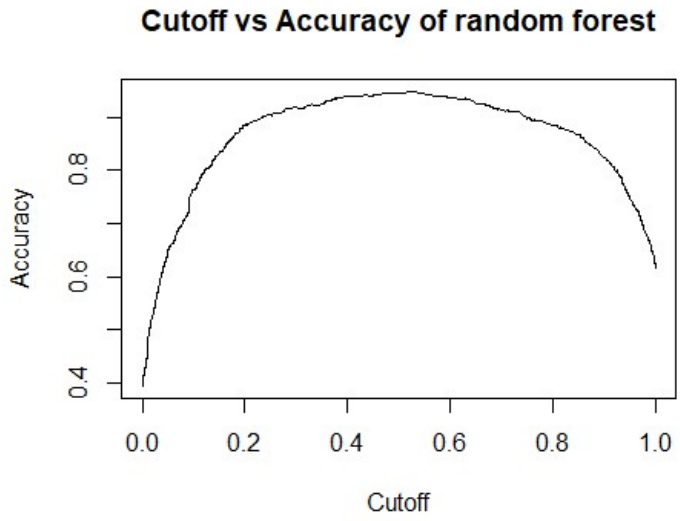
3-2. support vector machine with rbf kernel



Validation AUC 는 0.9730314, cutoff 가 0.5292952 일 때 Accuracy 가 0.9319319 로 최댓값을 가집니다. 위의 그래프를 보면 cutoff 에 대한 Accuracy 의 민감함이 glmnet 보다 덜 민감한 것처럼 보입니다. AUC 와 Accuracy 가 glmnet 보다 높은 수치를 기록하여, 성능이 더 좋은 모델이라고 평가할 수 있습니다.

```
##  AUC of svm : 0.9730314
##  Max Accuracy of svm : 0.9319319
##  Cutoff of maximum accuracy of svm : 0.5292952
```

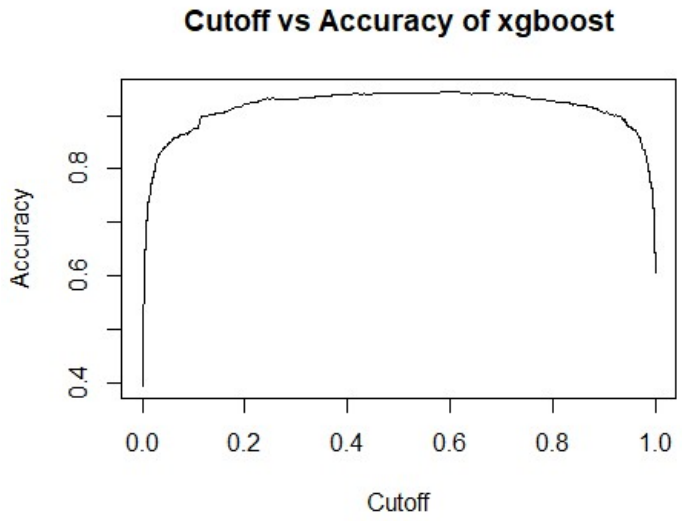
3-3. random forest



Validation AUC 는 0.9830914, cutoff 가 0.5272259 일 때 Accuracy 가 0.9479479 로 최댓값을 가집니다. 4 개의 모델 중 가장 높은 Accuracy 를 가지지만 AUC 는 xgboost 보다 살짝 낮은 성능을 보여줍니다. 즉 Accuracy 가 기준일 땐 random forest 가 가장 좋은 모델로 평가할 수 있지만 AUC 기준일 땐 2 번째로 좋은 모델이라고 평가할 수 있습니다다.

```
##  AUC of rf : 0.9830914
##  Max Accuracy of rf : 0.9479479
##  Cutoff of maximum accuracy of rf : 0.5272259
```

3-4. xgboost

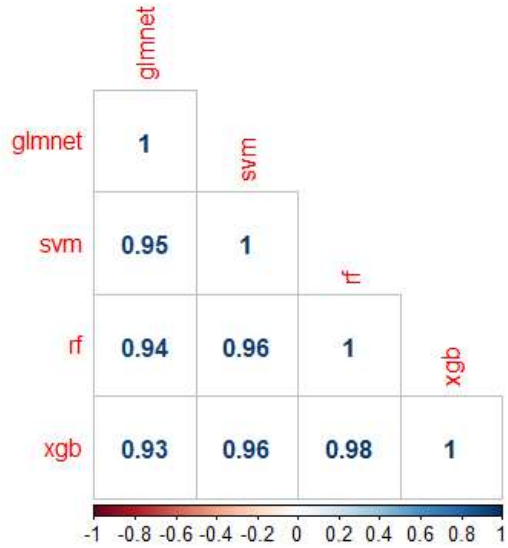


Validation AUC 는 0.9832278, cutoff 가 0.5992255 일 때 Accuracy 가 0.9449449 로 최댓값을 가집니다. 4 개의 모델 중 가장 높은 AUC 를 가지지만 Accuracy 는 random forest 보다 살짝 낮은 성능을 보여줍니다. 즉 AUC 가 기준일 땐 xgboost 가 가장 좋은 모델로 평가할 수 있지만 Accuracy 가 기준일 땐 2 번째로 좋은 모델이라고 평가할 수 있습니다.

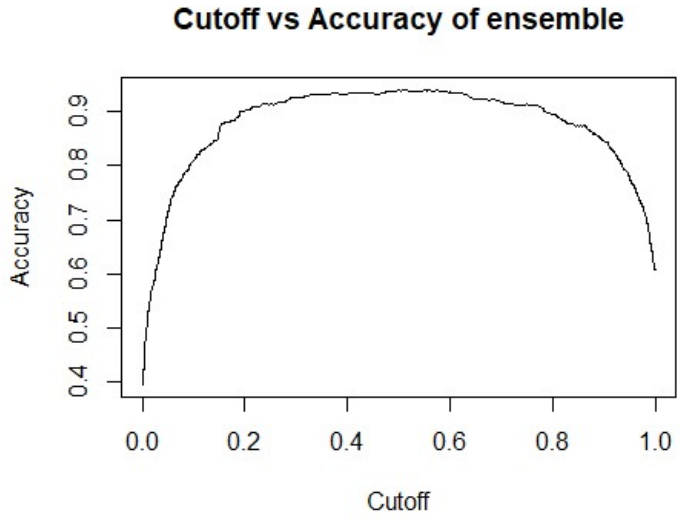
```
##  AUC of xgb : 0.9832278
##  Max Accuracy of xgb : 0.9449449
##  Cutoff of maximum accuracy of xgb : 0.5992255
```

기준 통계량이 AUC 인지 Accuracy 인지에 따라 가장 좋은 모델이 달라지므로, 일반화가 더 잘되는 모델을 구하기 위해선 위에서 구한 4 개의 모델을 앙상블하는 방법을 고려해볼 수 있다고 생각합니다.

4. choose models to use or to make ensemble



4 개의 validation posterior probability 들이 강한 상관성을 가지고 있어서 앙상블 했을 때 성능이 훨씬 더 좋아지지는 않을 것으로 추정됩니다. 하지만 앞서 언급했듯이 기준 통계량에 따라 가장 좋은 모델이 바뀌므로 4 개의 모델을 앙상블하여 최종 예측값을 구하려고 합니다.



앙상블을 했을 때 validation AUC 는 0.9814973, cutoff 가 0.516017 일 때 maximum Accuracy 는 0.9409490 입니다. 개별 모델들과 비교했을 때 random forest, xgboost 보다 AUC 가 조금 낮아졌지만 대신 결과를 합했기 때문에 더 견고한 모델을 얻었다고 평가할 수 있습니다. 이 모델을 최종 모델로 사용하기로 했고 test class 를 예측할 때의 cutoff 는 ensemble 의 cutoff 인 0.516017 로 사용하기로 결정했습니다.

```
##  AUC of ensemble : 0.9814973
##  Max Accuracy of ensemble : 0.9409409
##  Cutoff of maximum accuracy of ensemble : 0.516017
```

```
##                AUC  Accuracy    Cutoff
## glmnet   0.9628162 0.9059059 0.3263440
## svm      0.9730314 0.9319319 0.5292952
## rf       0.9830914 0.9479479 0.5272259
## xgb      0.9832278 0.9449449 0.5992255
## ensemble 0.9814973 0.9409409 0.5160170
```

5. confime final model and predict class and posterior probability of Xtest

4000 개의 train 데이터 전부를 이용하여 glmnet, svmradial, random forest, xgboost 를 학습했습니다. 그때의 하이퍼 파라미터는 3001 개의 데이터만 사용해서 구한 최적 하이퍼 파라미터를 사용하였습니다. Test 데이터의 class '1'에 대한 4 개의 posterior probability 를 평균 내어 최종 ensemble posterior probability 를 구할 수 있었습니다. cutoff 는 validation ensemble accuracy 를 최대로 하는 0.516017 를 사용하였고, 1 일 확률이 cutoff 보다 크면 1 로 예측하고 cutoff 보다 작으면 0 으로 예측하여 최종 predicted test class 를 얻을 수 있었습니다.

Appendix : R code

library package to use

```
library(caret)
library(dplyr)
library(ROCR) # for auc, acc, cutoff
library(glmnet)
library(kernlab) # for support vector machine
library(ranger) # for random forest as ranger
library(xgboost) # for xgboost
library(writexl) # for exporting answer
library(ggplot2)
library(corrplot)
```

1. load and split dataset

load dataset, split dataset as train and validation using stratifed partition based on target variable

```
trainset = read.csv('./Train.csv', header=TRUE)
testset = read.csv('./Xtest.csv', header=TRUE)

train_X = trainset[, 2:51]
train_Y = as.factor(trainset[, 52])
test_X = testset[, 2:51]




student = 20152410
train_size = 0.75
set.seed(student)
train_index = createDataPartition(train_Y, p=train_size)

X_train = train_X[train_index$Resample1, ]
Y_train = train_Y[train_index$Resample1]
X_val = train_X[-train_index$Resample1, ]
Y_val = train_Y[-train_index$Resample1]

factor_Y_train = ifelse(Y_train == '1', 'yes', 'no')

# check wheter data is stratifed based on target

cat(' ratio of trainset target :', summary(train_Y)[1] / summary(train_Y)[2],
    '\n ratio of train target :', summary(Y_train)[1] / summary(Y_train)[2],
    '\n ratio of validation target :', summary(Y_val)[1] / summary(Y_val)[2])
```

```
##  ratio of trainset target : 1.533249
##  ratio of train target : 1.532489
##  ratio of validation target : 1.535533
```

2. data modeling

data modeling using caret to tune hyper parameter of candidate models

candidate : glmnet, svmradial, random forest, xgboost

2-1. glmnet

```
glmnet_tune_length = 250
glmnet_fold_number = 4
glmnet_train_control = trainControl(method='cv',
                                    number=glmnet_fold_number,
                                    search='random',
                                    classProbs = TRUE,
                                    summaryFunction=twoClassSummary)

glmnet_start = Sys.time()

set.seed(student)
glmnet_model = train(X_train,
                     factor_Y_train,
                     method='glmnet',
                     trControl=glmnet_train_control,
                     metric='ROC',
                     tuneLength=glmnet_tune_length,
                     preProcess = c('center', 'scale'))

glmnet_train_time = Sys.time() - glmnet_start
cat('Train time of glmnet model : '); glmnet_train_time
```

```
## Train time of glmnet model :
```

```
## Time difference of 13.29373 mins
```

```
best_alpha = as.numeric(glmnet_model$bestTune[1])
best_lambda = as.numeric(glmnet_model$bestTune[2])

glmnet_pred = predict(glmnet_model, newdata=X_val, type='prob')

glmnet_best_tune = glmnet_model$results %>%
  arrange(desc(ROC)) %>%
  head(1)
glmnet_best_tune
```

```
##       alpha      lambda       ROC      Sens     Spec       ROCSD     SensSD
## 1 0.9571266 0.001491034 0.9615987 0.9449339 0.861586 0.009071119 0.02112701
##       SpecSD
## 1 0.02058914
```

2-2. support vector machine with rbf kernel

```
svm_tune_length = 250
svm_fold_number = 4
svm_train_control = trainControl(method='cv',
                                 number=svm_fold_number,
                                 search='random',
                                 classProbs = TRUE,
                                 summaryFunction=twoClassSummary)

svm_start = Sys.time()

set.seed(student)
svm_model = train(X_train,
                  factor_Y_train,
                  method='svmRadial',
                  trControl=svm_train_control,
                  metric='ROC',
                  tuneLength=svm_tune_length,
                  preProcess = c('center', 'scale'))

svm_train_time = Sys.time() - svm_start
cat('Train time of svm model : '); svm_train_time
```

```
## Train time of svm model :
```

```
## Time difference of 38.41799 mins
```

```
best_sigma = as.numeric(svm_model$bestTune[1])
best_C = as.numeric(svm_model$bestTune[2])

svm_pred = predict(svm_model, newdata=X_val, type='prob')

svm_best_tune = svm_model$results %>%
  arrange(desc(ROC)) %>%
  head(1)
svm_best_tune
```

```
##         sigma        C       ROC      Sens      Spec       ROCSD    SensSD
## 1 0.009966466 21.71236 0.9678749 0.9526432 0.8801557 0.003811696 0.0137555
##       SpecSD
## 1 0.01510196
```

2-3. random forest

```
rf_tune_length = 250
rf_fold_number = 4
rf_train_control = trainControl(method='cv',
                                number=rf_fold_number,
                                search='random',
                                classProbs=TRUE,
                                summaryFunction=twoClassSummary)

rf_start = Sys.time()

set.seed(student)
rf_model = train(X_train,
                 factor_Y_train,
                 method='ranger',
                 trControl=rf_train_control,
                 metric='ROC',
                 tuneLength=rf_tune_length)

rf_train_time = Sys.time() - rf_start
cat('Train time of random forest model : '); rf_train_time
```

```
## Train time of random forest model :
```

```
## Time difference of 41.18659 mins
```

```
best_mtry = as.numeric(rf_model$bestTune[1])
best_splitrule = as.character(rf_model$bestTune[2][1, 1])
best_min.node.size = as.numeric(rf_model$bestTune[3])

rf_pred = predict(rf_model, newdata=X_val, type='prob')

rf_best_tune = rf_model$results %>%
  arrange(desc(ROC)) %>%
  head(1)
rf_best_tune
```

```
##   min.node.size mtry splitrule      ROC      Sens      Spec      ROCSD
## 1             4    4      gini 0.977648 0.9609031 0.8919687 0.00258328
##      SensSD     SpecSD
## 1 0.0136226 0.01978735
```

2-4. xgboost

```
xgb_tune_length = 250
xgb_fold_number = 4
xgb_train_control = trainControl(method='cv',
                                 number=xgb_fold_number,
                                 search='random',
                                 classProbs=TRUE,
                                 summaryFunction=twoClassSummary)

xgb_start = Sys.time()

set.seed(student)
xgb_model = train(X_train,
                  factor_Y_train,
                  method='xgbTree',
                  trControl=xgb_train_control,
                  metric='ROC',
                  tuneLength=xgb_tune_length)

xgb_train_time = Sys.time() - xgb_start
cat('Train time of xgboost model : '); xgb_train_time
```

```
## Train time of xgboost model :
```

```
## Time difference of 37.58226 mins
```

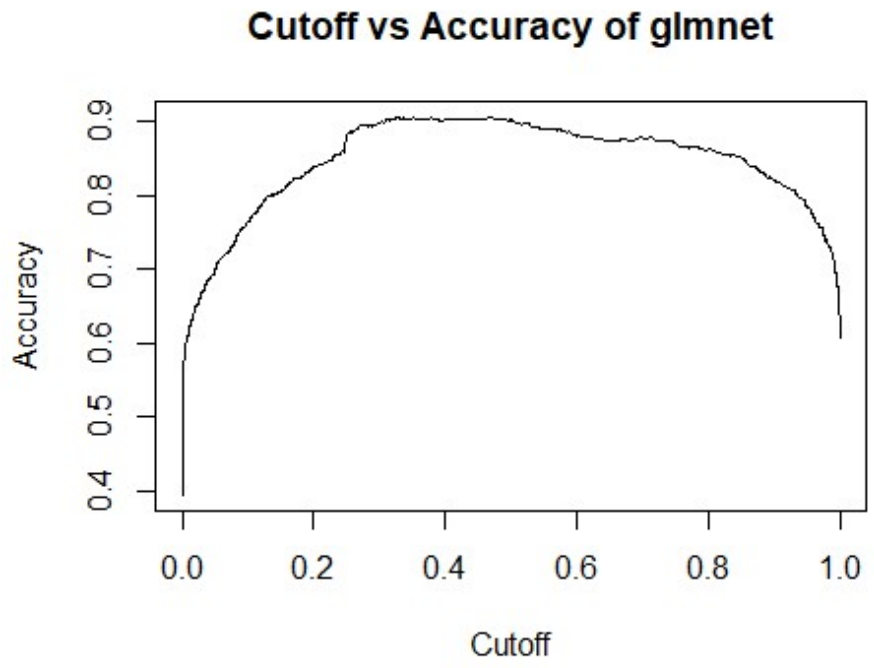
```
best_nrounds = as.numeric(xgb_model$bestTune[1])
best_max_depth = as.numeric(xgb_model$bestTune[2])
best_eta = as.numeric(xgb_model$bestTune[3])
best_gamma = as.numeric(xgb_model$bestTune[4])
best_colsample_bytree = as.numeric(xgb_model$bestTune[5])
best_min_child_weight = as.numeric(xgb_model$bestTune[6])
best_subsample = as.numeric(xgb_model$bestTune[7])

xgb_pred = predict(xgb_model, newdata=X_val, type='prob')

xgb_best_tune = xgb_model$results %>%
  arrange(desc(ROC)) %>%
  head(1)
xgb_best_tune
```

```
##         eta max_depth    gamma colsample_bytree min_child_weight subsample
## 1 0.09257654         9 1.453736        0.3281601                0 0.9524782
##   nrounds       ROC      Sens     Spec       ROCSD   SensSD     SpecSD
## 1     338 0.9800166 0.9548458 0.902937 0.005578064 0.014218 0.01374313
```

3. evaluate the models

3-1. glmnet

```
glmnet_prediction = prediction(glmnet_pred['yes'], Y_val)
glmnet_performance_auc = performance(glmnet_prediction, 'auc', 'cutoff')
glmnet_performance_acc = performance(glmnet_prediction, 'acc', 'cutoff')
plot(glmnet_performance_acc, main='Cutoff vs Accuracy of glmnet')
```

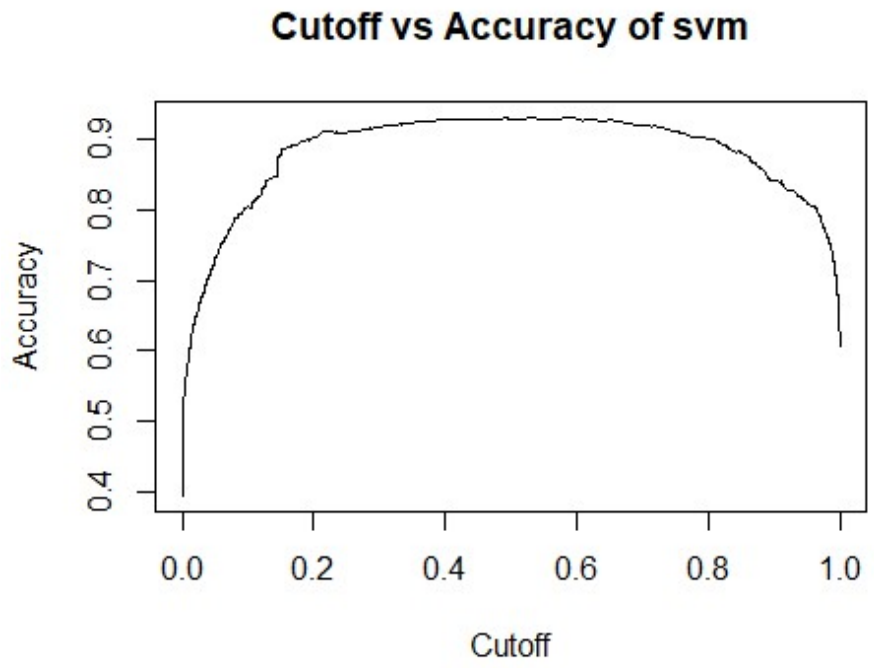



```
glmnet_auc = glmnet_performance_auc@y.values[[1]]
glmnet_acc = max(glmnet_performance_acc@y.values[[1]])
glmnet_cutoff = glmnet_performance_acc@x.values[[1]][which.max(glmnet_performanc
e_acc@y.values[[1]])]

cat(' AUC of glmnet :', glmnet_auc,
    '\n Max Accuracy of glmnet :', glmnet_acc,
    '\n Cutoff of maximum accuracy of glmnet :', glmnet_cutoff)
```

```
##  AUC of glmnet : 0.9628162
##  Max Accuracy of glmnet : 0.9059059
##  Cutoff of maximum accuracy of glmnet : 0.326344
```

3-2. support vector machine with rbf kernel

```
svm_prediction = prediction(svm_pred['yes'], Y_val)
svm_performance_auc = performance(svm_prediction, 'auc', 'cutoff')
svm_performance_acc = performance(svm_prediction, 'acc', 'cutoff')
plot(svm_performance_acc, main='Cutoff vs Accuracy of svm')
```

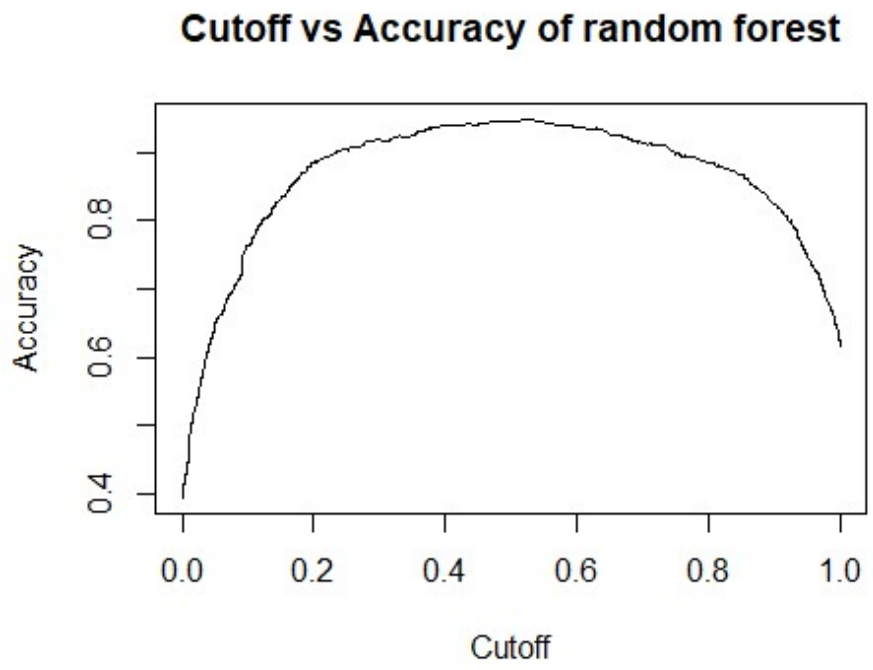



```
svm_auc = svm_performance_auc@y.values[[1]]
svm_acc = max(svm_performance_acc@y.values[[1]])
svm_cutoff = svm_performance_acc@x.values[[1]][which.max(svm_performance_acc@y.v
alues[[1]])]

cat(' AUC of svm :', svm_auc,
    '\n Max Accuracy of svm :', svm_acc,
    '\n Cutoff of maximum accuracy of svm :', svm_cutoff)
```

```
##  AUC of svm : 0.9730314
##  Max Accuracy of svm : 0.9319319
##  Cutoff of maximum accuracy of svm : 0.5292952
```

3-3. random forest

```
rf_prediction = prediction(rf_pred['yes'], Y_val)
rf_performance_auc = performance(rf_prediction, 'auc', 'cutoff')
rf_performance_acc = performance(rf_prediction, 'acc', 'cutoff')
plot(rf_performance_acc, main='Cutoff vs Accuracy of random forest')
```

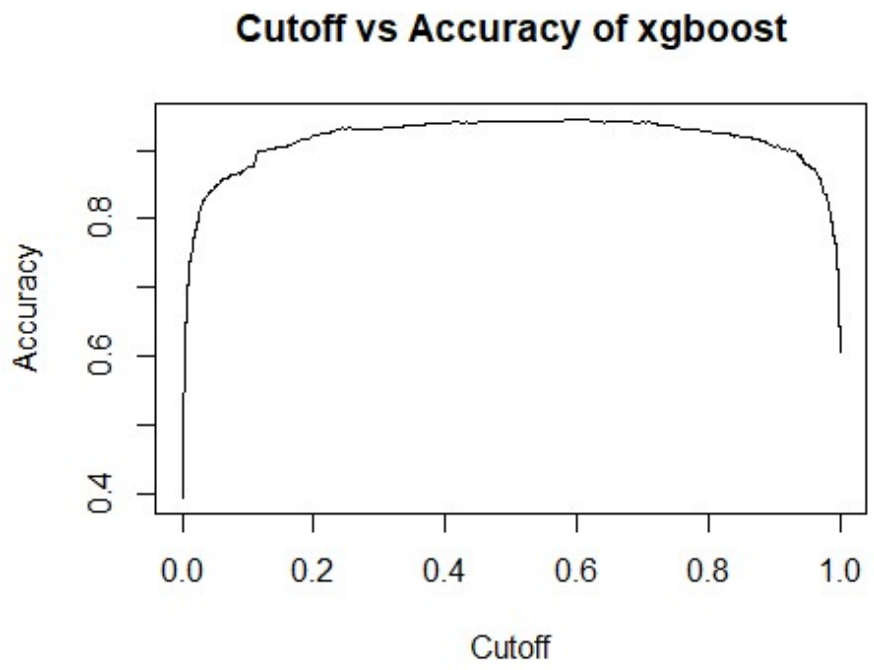



```
rf_auc = rf_performance_auc@y.values[[1]]
rf_acc = max(rf_performance_acc@y.values[[1]])
rf_cutoff = rf_performance_acc@x.values[[1]][which.max(rf_performance_acc@y.valu
es[[1]])]

cat(' AUC of rf :', rf_auc,
    '\n Max Accuracy of rf :', rf_acc,
    '\n Cutoff of maximum accuracy of rf :', rf_cutoff)
```

```
##  AUC of rf : 0.9830914
##  Max Accuracy of rf : 0.9479479
##  Cutoff of maximum accuracy of rf : 0.5272259
```

3-4. xgboost

```
xgb_prediction = prediction(xgb_pred['yes'], Y_val)
xgb_performance_auc = performance(xgb_prediction, 'auc', 'cutoff')
xgb_performance_acc = performance(xgb_prediction, 'acc', 'cutoff')
plot(xgb_performance_acc, main='Cutoff vs Accuracy of xgboost')
```

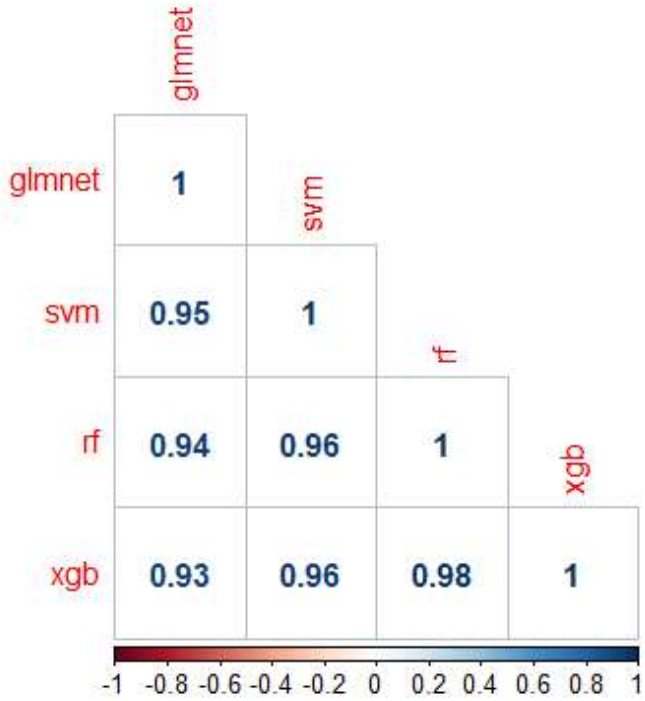



```
xgb_auc = xgb_performance_auc@y.values[[1]]
xgb_acc = max(xgb_performance_acc@y.values[[1]])
xgb_cutoff = xgb_performance_acc@x.values[[1]][which.max(xgb_performance_acc@y.v
alues[[1]])]

cat(' AUC of xgb :', xgb_auc,
    '\n Max Accuracy of xgb :', xgb_acc,
    '\n Cutoff of maximum accuracy of xgb :', xgb_cutoff)
```

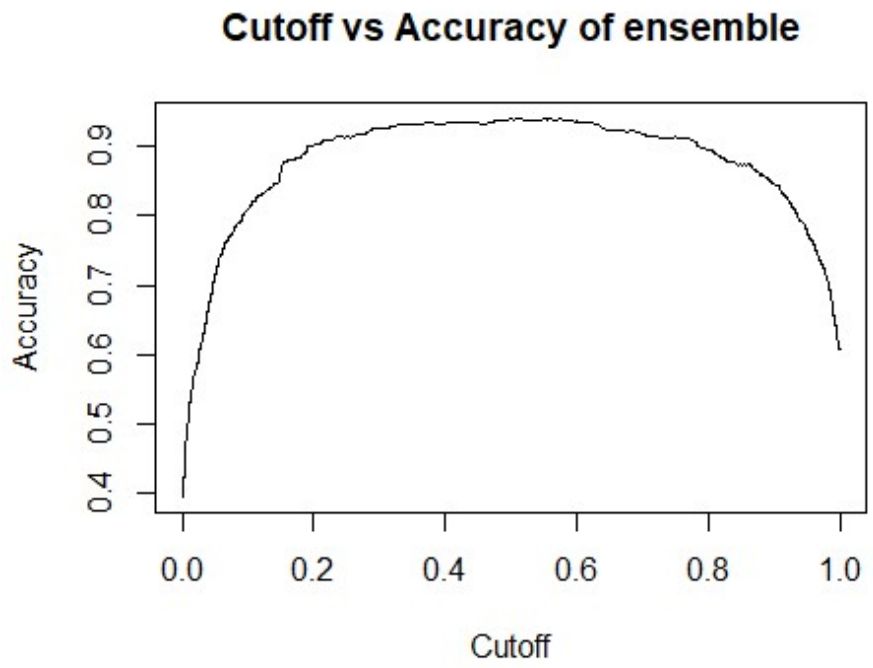
```
##  AUC of xgb : 0.9832278
##  Max Accuracy of xgb : 0.9449449
##  Cutoff of maximum accuracy of xgb : 0.5992255
```

4. choose models to use or to make ensemble

```
validation_pred = data.frame(glmnet_pred['yes'],
                             svm_pred['yes'],
                             rf_pred['yes'],
                             xgb_pred['yes'])
colnames(validation_pred) = c('glmnet', 'svm', 'rf', 'xgb')

corr_matrix = cor(validation_pred)
corrplot(corr_matrix, method='number', type='lower')
```




```
validation_pred = validation_pred %>%
  mutate(ensemble = (glmnet + svm + rf + xgb) / 4)

ensemble_prediction = prediction(validation_pred$ensemble, Y_val)
ensemble_performance_auc = performance(ensemble_prediction, 'auc', 'cutoff')
ensemble_performance_acc = performance(ensemble_prediction, 'acc', 'cutoff')
plot(ensemble_performance_acc, main='Cutoff vs Accuracy of ensemble')
```




```
ensemble_auc = ensemble_performance_auc@y.values[[1]]
ensemble_acc = max(ensemble_performance_acc@y.values[[1]])
ensemble_cutoff = ensemble_performance_acc@x.values[[1]][which.max(ensemble_perf
ormance_acc@y.values[[1]])]

cat(' AUC of ensemble :', ensemble_auc,
    '\n Max Accuracy of ensemble :', ensemble_acc,
    '\n Cutoff of maximum accuracy of ensemble :', ensemble_cutoff)
```

```
##  AUC of ensemble : 0.9814973
##  Max Accuracy of ensemble : 0.9409409
##  Cutoff of maximum accuracy of ensemble : 0.516017
```

```
modeling_result = data.frame(AUC = c(glmnet_auc, svm_auc, rf_auc, xgb_auc, ensemb
le_auc),
                             Accuracy = c(glmnet_acc, svm_acc, rf_acc, xgb_acc, ens
emble_acc),
                             Cutoff = c(glmnet_cutoff, svm_cutoff, rf_cutoff, xgb_c
utoff, ensemble_cutoff))
rownames(modeling_result) = c('glmnet', 'svm', 'rf', 'xgb', 'ensemble')
modeling_result
```

```
##                AUC  Accuracy    Cutoff
## glmnet   0.9628162 0.9059059 0.3263440
## svm      0.9730314 0.9319319 0.5292952
## rf       0.9830914 0.9479479 0.5272259
## xgb      0.9832278 0.9449449 0.5992255
## ensemble 0.9814973 0.9409409 0.5160170
```

5. confime final model and predict class and posterior probability of Xtest

5-1. glmnet

```
final_glmnet_model = glmnet(x=as.matrix(train_X),
                            y=train_Y,
                            family='binomial',
                            alpha=best_alpha,
                            lambda=best_lambda,
                            standardize = TRUE)

final_glmnet_pred = predict(final_glmnet_model, newx=as.matrix(test_X), type='re
sponse')[, 1]

final_glmnet_model
```

```
##
## Call:  glmnet(x = as.matrix(train_X), y = train_Y, family = "binomial",      a
lpha = best_alpha, lambda = best_lambda, standardize = TRUE)
##
##      Df   %Dev   Lambda
## [1,] 47 0.6367 0.001491
```

5-2. support vector machine with rbf kernel

```
final_svm_model = ksvm(train_Y ~ .,
                       data=cbind(train_X, train_Y),
                       scaled=TRUE,
                       type='C-svc',
                       kernel='rbfdot',
                       kpar=list(sigma=best_sigma),
                       C=best_C,
                       prob.model=TRUE)

final_svm_pred = predict(final_svm_model, newdata=test_X, type="probabilities")
[, 2]

final_svm_model
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 21.7123636978789
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.00996646616396962
##
## Number of Support Vectors : 896
##
## Objective Function Value : -12279.39
## Training error : 0.04625
## Probability model included.
```

5-3. random forest

```
final_rf_model = ranger(train_Y ~ .,
                        data=cbind(train_X, train_Y),
                        mtry=best_mtry,
                        splitrule=best_splitrule,
                        min.node.size=best_min.node.size,
                        probability=TRUE)

final_rf_pred_ = predict(final_rf_model, data=test_X, type="response",
                         num.trees=final_rf_model$num.trees)
final_rf_pred = final_rf_pred_$predictions[, 2]

final_rf_model
```

```
## Ranger result
##
## Call:
##  ranger(train_Y ~ ., data = cbind(train_X, train_Y), mtry = best_mtry,      sp
litrule = best_splitrule, min.node.size = best_min.node.size,      probability =
TRUE)
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      4000
## Number of independent variables:  50
## Mtry:                             4
## Target node size:                 4
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.05036759
```

5-4. xgboost

```
temp_train_Y = ifelse(train_Y == '1', 1, 0)
final_xgb_model = xgboost(data=as.matrix(train_X),
                          label=temp_train_Y,
                          objective='binary:logistic',
                          nrounds=best_nrounds,
                          max_depth=best_max_depth,
                          eta=best_eta,
                          gamma=best_gamma,
                          colsample_bytree=best_colsample_bytree,
                          min_child_weight=best_min_child_weight,
                          subsample=best_subsample,
                          verbose=FALSE)

final_xgb_pred = predict(final_xgb_model, newdata=as.matrix(test_X), type="prob
")

final_xgb_model
```

```
## ##### xgb.Booster
## raw: 1.3 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, objective = "binary:logistic", max_depth = ..2,
##     eta = ..3, gamma = ..4, colsample_bytree = ..5, min_child_weight = ..6,
##     subsample = ..7)
## params (as set within xgb.train):
##   objective = "binary:logistic", max_depth = "9", eta = "0.0925765427979641",
gamma = "1.45373629638925", colsample_bytree = "0.328160088695586", min_child_we
ight = "0", subsample = "0.95247815316543", silent = "1"
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 50
## niter: 338
## nfeatures : 50
## evaluation_log:
##     iter train_error
##        1     0.11950
##        2     0.08475
## ---
##      337     0.01750
##      338     0.01750
```

5-5. ensemble

```
final_ensemble_pred = (final_glmnet_pred + final_svm_pred + final_rf_pred + final
_xgb_pred) / 4
final_ensemble_class = ifelse(final_ensemble_pred >= ensemble_cutoff, '1', '0')

final_ensemble_ = data.frame(final_ensemble_class, final_ensemble_pred)
final_ensemble = cbind(rownames(final_ensemble_), final_ensemble_)
colnames(final_ensemble) = c('ID', 'yhat', 'prob')

head(final_ensemble)
```

```
##   ID yhat      prob
## 1  1    1 0.99590291
## 2  2    0 0.28749520
## 3  3    0 0.01066789
## 4  4    1 0.61025069
## 5  5    1 0.94874179
## 6  6    1 0.75386461
```

```
write_xlsx(final_ensemble, path='./final_answer.xlsx')
```