stove - classification

2022.08.24.

Table of contents

1 Introduction

1) 본 문서는 stove 패키지를 Shiny app에서 사용하는 것을 상정해 작성했습니다.

2) 본 문서의 케이스 스타일은 Camel case와 Snake case가 혼용되어 있습니다.

- Camel case : stove의 함수명 및 파라미터명
- Snake case: 유저로부터 받는 입력, shiny app의 server에서 사용(될 것이라고 예상)하는 object명, snake case로 작성된 dependencies의 함수명 등

2 Import sample data

1) 전처리가 완료된 샘플데이터를 불러옵니다.

- NA가 없어야 함
- string value가 있는 열은 factor로 변환
- 한 열이 모두 같은 값으로 채워져 있을 경우 제외해야 함
- Date type column이 없어야 함
- Outcome 변수는 classification의 경우 factor, regression의 경우 numeric이어야 함 (clustering은 outcome변수를 사용하지 않음)

2) 본 문서에서 사용한 혈액검사 샘플데이터의 정보는 아래와 같습니다.

- SEX : 성별(남성:1, 여성:2)
- AGE_G : 연령(그룹)
- HGB : 혈색소
- TCHOL : 총콜레스테롤
- TG : 중성지방
- HDL : HDL 콜레스테롤
- ANE : 빈혈 진료여부(있음:1, 없음:0)
- IHD : 허혈심장질환 진료여부(있음:1, 없음:0)
- STK : 뇌혈관질환 진료여부(있음:1, 없음:0)

3) N수가 너무 크면 알고리즘에 따라 모델링 시간이 너무 길어질 수 있습니다. 빠른 결과를 얻고싶다면 1,000개 이하로 sampling하는 것을 추천합니다.

```
# remotes::install_github("statgarten/datatoys")
library(stove)
library(datatoys)
library(dplyr)

set.seed(1234)

cleaned_data <- datatoys::bloodTest

cleaned_data <- cleaned_data %>%
  mutate_at(vars(SEX, ANE, IHD, STK), factor) %>%
  mutate(TG = ifelse(TG < 150, 0, 1)) %>%
```

```
  mutate_at(vars(TG), factor) %>%
  group_by(TG) %>%
  sample_n(500) # TG(0):TG(1) = 500:500
```

3 Data Setup Tab

User Input	description
target_var	목적 변수
train_set_ratio	전체 데이터 중 train set의 비율 (range: 0.0 - 1.0)

1) User input을 다음과 같이 받습니다.

- formula는 user가 target_var를 입력할 때 함께 생성되도록 함

```
target_var <- "TG"
train_set_ratio <- 0.7
seed <- 1234
formula <- paste0(target_var, " ~ .")
```

2) Train-test split 작업이 완료된 Object를 저장하고, Train set을 보여줍니다.

```
split_tmp <- stove::trainTestSplit(data = cleaned_data,
                                   target = target_var,
                                   prop = train_set_ratio,
                                   seed = seed
                                   )

data_train <- split_tmp[[1]] # train data
data_test <- split_tmp[[2]] # test data
data_split <- split_tmp[[3]] # whole data with split information
```

3) train set에 적용할 전처리 정보를 담은 recipe를 생성합니다

```
rec <- stove::prepForCV(data = data_train,
                        formula = formula,
                        seed = seed
                        )
```

4 Modeling Tab

User Input	description
algo	ML 알고리즘 선택
engine	engine 선택
mode	mode 선택
metric	Best performance에 대한 평가지표 선택
v	Cross validation시 train set을 몇 번 분할할 것인지 입력
...	각 모델의 hyperparameter의 최소/최대값(Min, Max), 몇 단계로 나눌지(Levels)

모델 object를 저장할 빈 리스트를 생성합니다.

```
models_list <- list()
```

4.1 Logistic Regression

```
# User input

mode <- "classification"
algo <- "logisticRegression"
engine <- "glmnet" # glmnet (default), glm, stan

penalty_range_min <- 0.001
penalty_range_max <- 1.0
penalty_range_levels <- 5
mixture_range_min <- 0.0
mixture_range_max <- 1.0
mixture_range_levels <- 5

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::logisticRegression(
  algo = algo,
```

```
  engine = engine,
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  v = v,
  penaltyRangeMin = penalty_range_min,
  penaltyRangeMax = penalty_range_max,
  penaltyRangeLevels = penalty_range_levels,
  mixtureRangeMin = mixture_range_min,
  mixtureRangeMax = mixture_range_max,
  mixtureRangeLevels = mixture_range_levels,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.2 K Nearest Neighbor

```
# User input

mode <- "classification"
algo <- "KNN"
engine <- "kknn" # kknn (defualt)

neighbors_range_min <- 1
neighbors_range_max <- 10
neighbors_range_levels <- 10

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::KNN(
  algo = algo,
  engine = engine,
```

```
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  v = v,
  neighborsRangeMin = neighbors_range_min,
  neighborsRangeMax = neighbors_range_max,
  neighborsRangeLevels = neighbors_range_levels,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.3 Naive Bayes

```
# User input

mode <- "classification"
algo <- "naiveBayes"
engine <- "klaR" # klaR (default), naivebayes

smoothness_range_min <- 0.5
smoothness_range_max <- 1.5
smoothness_range_levels <- 3
laplace_range_min <- 0.0
laplace_range_max <- 3.0
laplace_range_levels <- 4

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::naiveBayes(
  algo = algo,
  engine = engine,
  mode = mode,
```

```
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  v = v,
  smoothnessRangeMin = smoothness_range_min,
  smoothnessRangeMax = smoothness_range_max,
  smoothnessRangeLevels = smoothness_range_levels,
  LaplaceRangeMin = laplace_range_min,
  LaplaceRangeMax = laplace_range_max,
  LaplaceRangeLevels = laplace_range_levels,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.4 Decision Tree

```
# User input

mode <- "classification"
algo <- "decisionTree"
engine <- "rpart" # rpart (default), C5.0, partykit

tree_depth_range_min <- 1
tree_depth_range_max <- 15
tree_depth_range_levels <- 3
min_n_range_min <- 2
min_n_range_max <- 40
min_n_range_levels <- 3
cost_complexity_range_min <- -2.0
cost_complexity_range_max <- -1.0
cost_complexity_range_levels <- 2

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling
```

```
finalized <- stove::decisionTree(
  algo = algo,
  engine = engine,
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  v = v,
  treeDepthRangeMin = tree_depth_range_min,
  treeDepthRangeMax = tree_depth_range_max,
  treeDepthRangeLevels = tree_depth_range_levels,
  minNRangeMin = min_n_range_min,
  minNRangeMax = min_n_range_max,
  minNRangeLevels = min_n_range_levels,
  costComplexityRangeMin = cost_complexity_range_min,
  costComplexityRangeMax = cost_complexity_range_max,
  costComplexityRangeLevels = cost_complexity_range_levels,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.5 Random Forest

```
# User input

mode <- "classification"
algo <- "randomForest"
engine <- "ranger" # ranger (default), randomForest, partykit

mtry_range_min <- 1
mtry_range_max <- 20
mtry_range_levels <- 3
trees_range_min <- 100
trees_range_max <- 1000
trees_range_levels <- 3
min_n_range_min <- 2
```

```r
min_n_range_max <- 40
min_n_range_levels <- 3

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::randomForest(
  algo = algo,
  engine = engine,
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  v = v,
  mtryRangeMin = mtry_range_min,
  mtryRangeMax = mtry_range_max,
  mtryRangeLevels = mtry_range_levels,
  treesRangeMin = trees_range_min,
  treesRangeMax = trees_range_max,
  treesRangeLevels = trees_range_levels,
  minNRangeMin = min_n_range_min,
  minNRangeMax = min_n_range_max,
  minNRangeLevels = min_n_range_levels,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.6 XGBoost

```r
# User input

mode <- "classification"
algo <- "XGBoost"
engine <- "xgboost" # xgboost
```

```r
tree_depth_range_min <- 5
tree_depth_range_max <- 15
tree_depth_range_levels <- 3
trees_range_min <- 8
trees_range_max <- 32
trees_range_levels <- 3
learn_rate_range_min <- -2.0
learn_rate_range_max <- -1.0
learn_rate_range_levels <- 2
mtry_range_min <- 0.0
mtry_range_max <- 1.0
mtry_range_levels <- 3
min_n_range_min <- 2
min_n_range_max <- 40
min_n_range_levels <- 3
loss_reduction_range_min <- -1.0
loss_reduction_range_max <- 1.0
loss_reduction_range_levels <- 3
sample_size_range_min <- 0.0
sample_size_range_max <- 1.0
sample_size_range_levels <- 3
stop_iter <- 30

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::xgBoost(
  algo = algo,
  engine = engine,
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  v = v,
  treeDepthRangeMin = tree_depth_range_min,
  treeDepthRangeMax = tree_depth_range_max,
  treeDepthRangeLevels = tree_depth_range_levels,
  treesRangeMin = trees_range_min,
```

```
  treesRangeMax = trees_range_max,
  treesRangeLevels = trees_range_levels,
  learnRateRangeMin = learn_rate_range_min,
  learnRateRangeMax = learn_rate_range_max,
  learnRateRangeLevels = learn_rate_range_levels,
  mtryRangeMin = mtry_range_min,
  mtryRangeMax = mtry_range_max,
  mtryRangeLevels = mtry_range_levels,
  minNRangeMin = min_n_range_min,
  minNRangeMax = min_n_range_max,
  minNRangeLevels = min_n_range_levels,
  lossReductionRangeMin = loss_reduction_range_min,
  lossReductionRangeMax = loss_reduction_range_max,
  lossReductionRangeLevels = loss_reduction_range_levels,
  sampleSizeRangeMin = sample_size_range_min,
  sampleSizeRangeMax = sample_size_range_max,
  sampleSizeRangeLevels = sample_size_range_levels,
  stopIter = stop_iter,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.7 lightGBM

```
# User input

mode <- "classification"
algo <- "lightGBM"
engine <- "lightgbm" # lightgbm

tree_depth_range_min <- 5
tree_depth_range_max <- 15
tree_depth_range_levels <- 3
trees_range_min <- 10
trees_range_max <- 100
trees_range_levels <- 2
learn_rate_range_min <- -2.0
learn_rate_range_max <- -1.0
```

```r
learn_rate_range_levels <- 2
mtry_range_min <- 1
mtry_range_max <- 20
mtry_range_levels <- 3
min_n_range_min <- 2
min_n_range_max <- 40
min_n_range_levels <- 3
loss_reduction_range_min <- -1.0
loss_reduction_range_max <- 1.0
loss_reduction_range_levels <- 3

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::lightGbm(
  algo = algo,
  engine = engine,
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  v = v,
  treeDepthRangeMin = tree_depth_range_min,
  treeDepthRangeMax = tree_depth_range_max,
  treeDepthRangeLevels = tree_depth_range_levels,
  treesRangeMin = trees_range_min,
  treesRangeMax = trees_range_max,
  treesRangeLevels = trees_range_levels,
  learnRateRangeMin = learn_rate_range_min,
  learnRateRangeMax = learn_rate_range_max,
  learnRateRangeLevels = learn_rate_range_levels,
  mtryRangeMin = mtry_range_min,
  mtryRangeMax = mtry_range_max,
  mtryRangeLevels = mtry_range_levels,
  minNRangeMin = min_n_range_min,
  minNRangeMax = min_n_range_max,
  minNRangeLevels = min_n_range_levels,
  lossReductionRangeMin = loss_reduction_range_min,
```

```
  lossReductionRangeMax = loss_reduction_range_max,
  lossReductionRangeLevels = loss_reduction_range_levels,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.8 MLP

```
# User input

mode <- "classification"
algo <- "MLP"
engine <- "nnet" # nnet

hidden_units_range_min <- 1
hidden_units_range_max <- 10
hidden_units_range_levels <- 3
penalty_range_min <- 0.001
penalty_range_max <- 1.0
penalty_range_levels <- 3
epochs_range_min <- 10
epochs_range_max <- 100
epochs_range_levels <- 2

v <- 2

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::MLP(
  algo = algo,
  engine = engine,
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
```

```
  v = v,
  hiddenUnitsRangeMin = hidden_units_range_min,
  hiddenUnitsRangeMax = hidden_units_range_max,
  hiddenUnitsRangeLevels = hidden_units_range_levels,
  penaltyRangeMin = penalty_range_min,
  penaltyRangeMax = penalty_range_max,
  penaltyRangeLevels = penalty_range_levels,
  epochsRangeMin = epochs_range_min,
  epochsRangeMax = epochs_range_max,
  epochsRangeLevels = epochs_range_levels,
  metric = metric
)

# Add the model to models_list
models_list[[paste0(algo, "_", engine)]] <- finalized$finalFittedModel
```

4.9 Modeling without hyperparameter

함수 내에 기본값을 선언해 뒀기때문에, 유저로부터 입력을 받지 않아도 모델링이 가능합니다. 아래처럼 hyperparameter관련 파라미터, v를 따로 입력받지 않아도 됩니다.

```
# User input

mode <- "classification"
algo <- "LogisticAuto"
engine <- "glmnet" # glmnet (default), glm, stan

metric <- "roc_auc" # roc_auc (default), accuracy

# Modeling

finalized <- stove::logisticRegression(
  algo = algo,
  engine = engine,
  mode = mode,
  trainingData = data_train,
  splitedData = data_split,
  formula = formula,
  rec = rec,
  # v = v,
  # penaltyRangeMin = penalty_range_min,
```

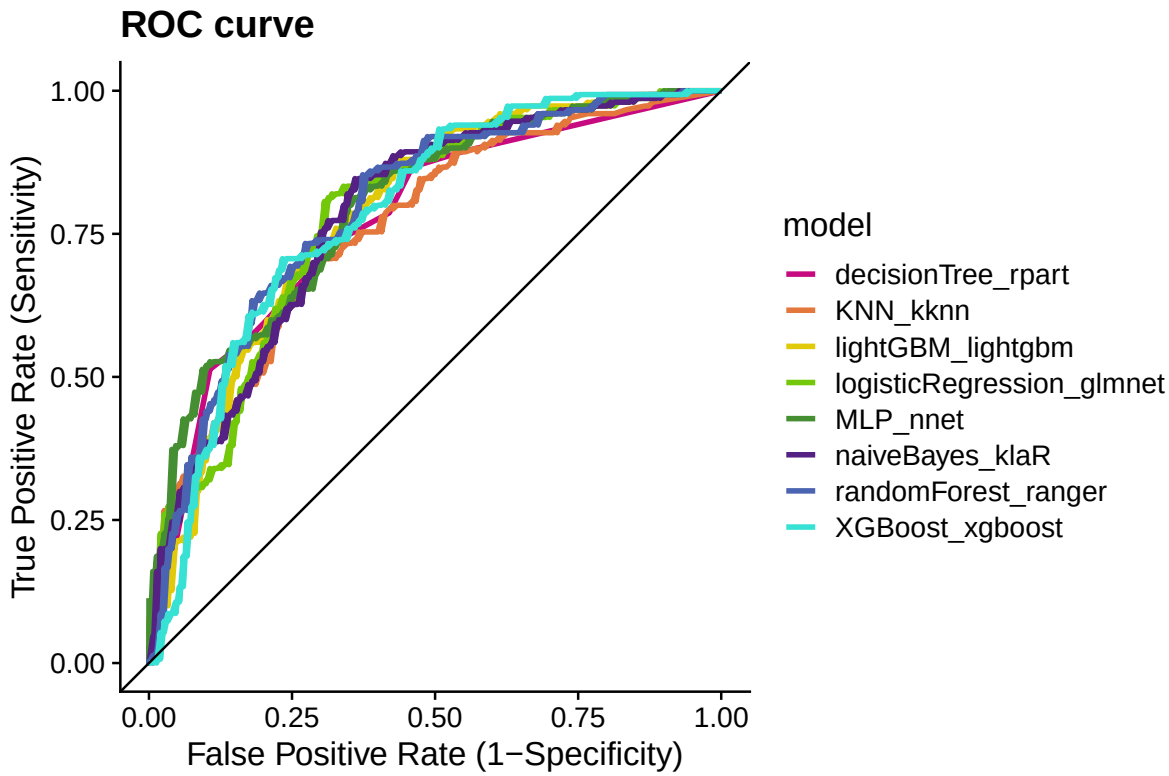
```
  # penaltyRangeMax = penalty_range_max,
  # penaltyRangeLevels = penalty_range_levels,
  # mixtureRangeMin = mixture_range_min,
  # mixtureRangeMax = mixture_range_max,
  # mixtureRangeLevels = mixture_range_levels,
  metric = metric
)
```

5 Sources for report

5.1 ROC Curve

유저가 선택한 모델의 ROC curve 출력

```
roc_curve <- stove::rocCurve(
  modelsList = models_list,
  targetVar = target_var
)
roc_curve
```



5.2 Confusion Matrix

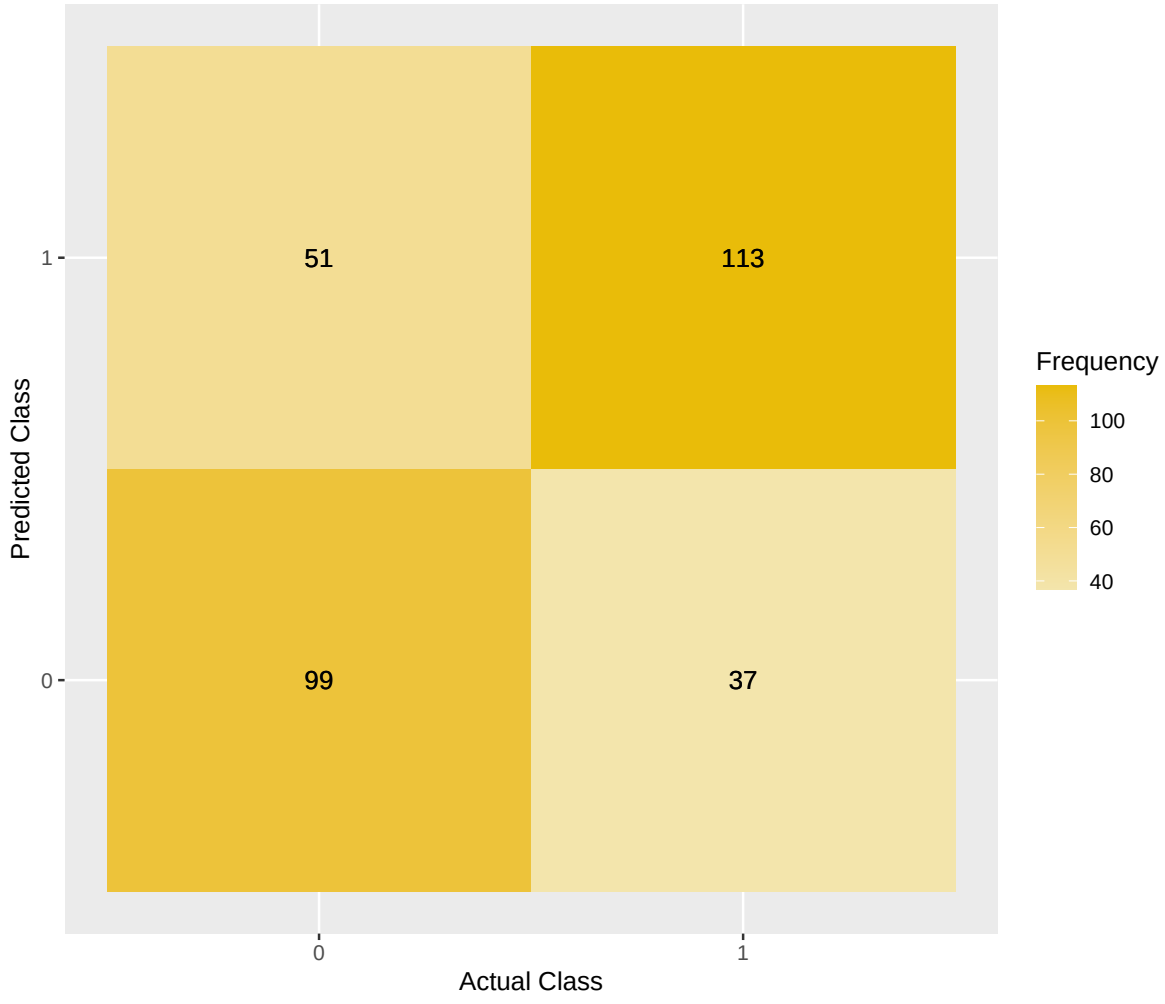
유저가 선택한 모델의 confusion matrix 출력 리스트 내 모델의 이름은 {algo}_{engine}의 형태로 저장되어 있음

```
# User input
names(models_list)
```

```
[1] "logisticRegression_glmnet" "KNN_kknn"
[3] "naiveBayes_klaR"           "decisionTree_rpart"
[5] "randomForest_ranger"       "XGBoost_xgboost"
[7] "lightGBM_lightgbm"         "MLP_nnet"
```

```
model_name <- "randomForest_ranger"
```

```
cm <- stove::confusionMatrix(
  modelName = model_name,
  modelsList = models_list,
  targetVar = target_var
)
cm
```



5.3 Evaluation metrics

- 모델 성능 비교를 위한 표 출력
- options(yardstick.event_level = “second”)은 오름차순으로 factor의 level 설정하기 위한 옵션

```
options(yardstick.event_level = "second")
evalMet <- stove::evalMetricsC(models_list, target_var)
knitr::kable(evalMet)
```

	Accuracy	Recall	Specificity	Precision	F1-score	Kappa	MCC
logisticRegression_glmnet	0.740	0.660	0.820	0.786	0.717	0.480	0.486
KNN_kknn	0.693	0.667	0.720	0.704	0.685	0.387	0.387
naiveBayes_klaR	0.703	0.713	0.693	0.699	0.706	0.407	0.407
decisionTree_rpart	0.703	0.667	0.740	0.719	0.692	0.407	0.408
randomForest_ranger	0.707	0.660	0.753	0.728	0.692	0.413	0.415
XGBoost_xgboost	0.720	0.733	0.707	0.714	0.724	0.440	0.440
lightGBM_lightgbm	0.710	0.687	0.733	0.720	0.703	0.420	0.420
MLP_nnet	0.703	0.673	0.733	0.716	0.694	0.407	0.407