Discrete Wavelet Transform을 이용한 Time Series Classification

방태모

예제. 분석에 사용된 데이터는 영국의 탄소 발자국(carbon foot print : 개인 또는 단체가 직접·간접적으로 발생시키는 온실 기체의 총량을 의미함. 여기에는 이들이 일상 생활에서 사용하는 연료, 전기, 용품 등이 모두 포함됨)을 줄이는 것에 도움을 주는 목적에서 수집되었고, 가정에서 전기를 어떻게 사용하고 있는지에 대한 행동(behavioral) 데이터에 해당한다. 각 series는 720의 길이(매 2분 마다 24시간 측정)를 가지고 있으며, CRT TV, LCD TV, 컴퓨터 모니터로 세 범주를 가진다.

```
> library(tidyverse)
> library(mlr)
> library(wavelets)
> library(caret)
> ## 데이터 로딩 및 전처리
> train_sctype <- read_table("./data/ScreenType/ScreenType_TRAIN.txt", col_names = FALSE) # 행 하나가 관측치
> test_sctype <- read_table("./data/ScreenType/ScreenType_TEST.txt", col_names = FALSE)
> train_sctype <- train_sctype %>%
+   mutate(type = factor(type, levels = c(1, 2, 3)))
> test_sctype <- test_sctype %>%
+   mutate(type = factor(type, levels = c(1, 2, 3)))

> ## wavelet transform 실시
> train2_sctype <- train_sctype %>%
+   select(-type)
> test2_sctype <- test_sctype %>%
+   select(-type)
> train_wt <- NULL
> test_wt <- NULL
> for(i in 1:nrow(train2_sctype)){
+   train <- t(train2_sctype[i, ])
+   test <- t(test2_sctype[i, ])
+   wt1 <- dwt(train, filter = "haar", boundary = "periodic")
+   wt2 <- dwt(test, filter = "haar", boundary = "periodic")
+   train_wt <- bind_rows(train_wt, unlist(list(W = wt1@W$W1, V = wt1@V$V1)))
+   test_wt <- bind_rows(test_wt, unlist(list(W = wt2@W$W1, V = wt2@V$V1)))
+ }
--------------------
```

(해설) wavelet transform에 의한 결과는 mother fucntion에 의한 것과 father function에 의한 것으로 나뉘어 진다. W는 mother function에 의한 것으로, wavelet coefficients라고도 하며 smoothing filter 역할을 한다. V는 father function에 의한 것으로, scaling coefficients라고도 하며 데이터가 어떻게 변화하는지(data's detail)에 대한 정보를 포함한다. 또한 원 계열의 절반 길이에 해당하는 vector만을 모형 적합에 사용할 예정이다. 다음은 Haar function의 mother function(ψ)과 father function(ϕ)에 해당한다.

$$\psi_{Haar}(t) = \begin{cases} -1 & 1/2 \le t < 1, \\ 1 & 0 \le t < 1/2, \\ 0 & \text{otherwise.} \end{cases}$$

$$\phi_{Haar}(t) = \begin{cases} 1 & 0 < t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

출처 : < Time Series Classification with Discrete Wavelet Transformed Data : Insights from an EmpIrical Study - Daoyuan Li >)

```
> train_wt <- bind_cols(train_sctype %>% select(type), train_wt)
> test_wt <- bind_cols(test_sctype %>% select(type), test_wt)

> ## 모형 적합 : Random forest, knn 이용
> train_Task <- makeClassifTask(data = train2_wt, target = "type")
> # define parameters for tuning
> rg_ps <- makeParamSet(
+   makeIntegerParam("num.trees", lower = 100, upper = 200), # 고려할 tree 총 갯수
+   makeIntegerParam("mtry", lower = 50, upper = 100), # 각 노드 분할시 고려하는 변
수의 갯수
+   makeIntegerParam("min.node.size", lower = 50, upper = 100) # 노드에 포함될 최소
표본 사이즈.
+ )
> knn_ps <- makeParamSet(
+   makeIntegerLearnerParam("k", lower = 50, upper = 120)
+ )
> # define search function
> rancontrol <- makeTuneControlRandom(maxit = 50L) # do 50 interations
> # set 5 fold cross validation
> set_cv <- makeResampleDesc("CV", iters = 5L)
> # tune parameters
> set.seed(124)
> rg_tune <- tuneParams(learner = rg, resampling = set_cv, task = train_Task,
+                       par.set = rg_ps, control = rancontrol, measures = acc)
[Tune] Started tuning learner classif.ranger for parameter set:
```

```
                  Type len Def      Constr Req Tunable Trafo
num.trees      integer   -    - 100 to 200   -     TRUE     -
mtry           integer   -    -  50 to 100   -     TRUE     -
min.node.size integer   -    -  50 to 100   -     TRUE     -
With control class: TuneControlRandom
Imputation value: -0
[Tune-x] 1: num.trees=133; mtry=85; min.node.size=84
[Tune-y] 1: acc.test.mean=0.4666667; time: 0.0 min
...
[Tune-x] 50: num.trees=142; mtry=98; min.node.size=80
[Tune-y] 50: acc.test.mean=0.4560000; time: 0.0 min
[Tune] Result: num.trees=194; mtry=87; min.node.size=70 : acc.test.mean=0.5066667

> knn_tune <- tuneParams(learner = knn, resampling = set_cv, task = train_Task,
+                        par.set = knn_ps, control = rancontrol, measures = acc)
[Tune] Started tuning learner classif.knn for parameter set:
     Type len Def     Constr Req Tunable Trafo
k integer   -    - 50 to 120   -     TRUE     -
With control class: TuneControlRandom
Imputation value: -0
[Tune-x] 1: k=73
[Tune-y] 1: acc.test.mean=0.3573333; time: 0.0 min
...
[Tune-x] 50: k=74
[Tune-y] 50: acc.test.mean=0.3626667; time: 0.0 min
[Tune] Result: k=89 : acc.test.mean=0.3840000

> # using hyperparameters for modeling
> rg_best <- setHyperPars(rg, par.vals = rg_tune$x)
> knn_best <- setHyperPars(knn, par.vals = knn_tune$x)

> # train a model
> mod_rg <- mlr::train(rg_best, train_Task)
> mod_knn <- mlr::train(knn_best, train_Task)
> getLearnerModel(mod_rg)
Ranger result

Call:
 ranger::ranger(formula = NULL, dependent.variable = tn, data = getTaskData(.task,
    .subset), probability = (.learner$predict.type == "prob"),          case.weights =
.weights, ...)
```

```
Type:                             Classification
Number of trees:                   194
Sample size:                      375
Number of independent variables:  716
Mtry:                             87
Target node size:                 70
Variable importance mode:          none
Splitrule:                        gini
OOB prediction error:              54.40 %

> getLearnerModel(mod_knn)$k
[1] 89

> # make predictions
> pred_test <- predict(mod_rg, newdata = test2_wt)
> confusionMatrix(pred_test$data$response, pred_test$data$truth)
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3
         1 66 54 53
         2 40 39 24
         3 19 32 48

Overall Statistics
                                          
               Accuracy : 0.408           
                 95% CI : (0.3578, 0.4596)
    No Information Rate : 0.3333          
    P-Value [Acc > NIR] : 0.0014867       
                                          
                  Kappa : 0.112           
 Mcnemar's Test P-Value : 0.0002389       

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.5280   0.3120   0.3840
Specificity            0.5720   0.7440   0.7960
Pos Pred Value         0.3815   0.3786   0.4848
Neg Pred Value         0.7079   0.6838   0.7210
```

```
Prevalence               0.3333    0.3333    0.3333
Detection Rate           0.1760    0.1040    0.1280
Detection Prevalence     0.4613    0.2747    0.2640
Balanced Accuracy        0.5500    0.5280    0.5900
> pred_test <- predict(mod_knn, newdata = test2_wt)
> confusionMatrix(pred_test$data$response, pred_test$data$truth)
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3
         1 75 79 78
         2 49 40 34
         3  1  6 13

Overall Statistics

               Accuracy : 0.3413
                 95% CI : (0.2934, 0.3918)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : 0.39

                  Kappa : 0.012

 Mcnemar's Test P-Value : <2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.6000   0.3200  0.10400
Specificity            0.3720   0.6680  0.97200
Pos Pred Value         0.3233   0.3252  0.65000
Neg Pred Value         0.6503   0.6627  0.68451
Prevalence             0.3333   0.3333  0.33333
Detection Rate         0.2000   0.1067  0.03467
Detection Prevalence   0.6187   0.3280  0.05333
Balanced Accuracy      0.4860   0.4940  0.53800
--------------------
```

(1) No Information Rate : 가장 많은 값이 발견된 분류의 비율

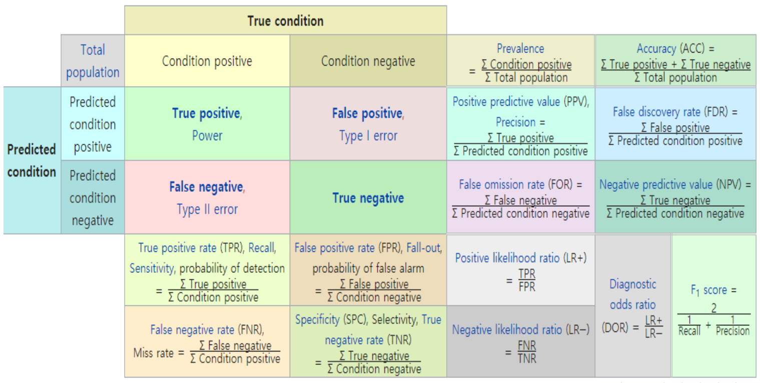
(2) Kappa : 범주형 자료에 대한 일치도 측도(measure of agreement)를 나타냄. 1은 완벽한 일치, 0은 일치 결여를 나타낸다. 군집이 심하게 불균형적인 경우 우수한 성능평가 측도.

$$K=\frac{Accuracy-p(e)}{1-p(e)}$$

$p(e)=\sum_{i=1}^{k} p_{i+}p_{+i}$: 우연히 실제값과 예측값이 일치된 평가를 할 비율

(3) McNemar`s Test P-Value : 동일한 개체에서 얻어진 두가지 범주형 변수의 반응률에 차이가 있는지 검정. (즉, paired t-test의 범주형 버전에 해당)

(4) Statistics by Class



		True condition				
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score $= \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$		

(출처 : 위키피디아)

① Sensitivty = Pr[Pred=Y|Ref=Y]

② Specificity = Pr[Pred=N|Ref=N]

③ Pos Pred Value(PPV) = Pr[Ref=Y|Pred=Y]

④ Neg Pred Value(NPV) = Pr[Ref=N|Pred=N]

⑤ Prevalence = Actual Yes/Total = $\frac{TP+FN}{TP+FP+FN+TN}$

⑥ Detection Rate = $\frac{TP}{TP+FP+FN+TN}$

⑦ Detection Prevalence = Pred Yes/Total = $\frac{TP+FP}{TP+FP+FN+TN}$

⑧ Balanced Accuracy = $(Sensitivity+Specificity)/2$