Data Augmentation using GAN or VAE

상세한 내용은 명한민논문토픽01.docx 문서를 참고합니다. (7월 이후 QSO 참고) 기존의 VAE, CVAE, GAN 모형들을 이용하여 data augmentation을 하는 경우 단순 복제와 차이가 없는 점에 착안하여 모형에 Prediction 모형을 포함하고 예측력을 loss함수에 추가하는 모형을 고려.

Data augmentation for sparse binary categorical response with VAE and Prediction Model

이를 위하여 로지스틱 회귀모형을 대상으로 여러가지 경우에 대한 비교분석을 수행

이 프로그램에서는 우선 downsampling 확률을 매우 작게 하여 random sampling 방법을 적용한 단순한 oversampling 기법을 사용하여 augmented 된 자료를 생성하고 훈련자료와 테스트 자료를 구분하며 준비.

VAE 모형에 가장 기본적인 FFN을 예측모형으로 추가하고 효과를 검토한다.

우선 기본적인 로지스틱 회귀모형을 가정하고 이를 따르는 자료를 생성

```
# fix the seed for comparison
```

VAE model Generation

```
# Let's GO!
if (tensorflow::tf$executing_eagerly())
  tensorflow::tf$compat$v1$disable_eager_execution()

library(keras)
library(keras)
library(caret)
```

```
## 필요한 패키지를 로딩중입니다: ggplot2
```

```
## 필요한 패키지를 로딩중입니다: lattice
```

```
library(InformationValue)
```

```
##
## 다음의 패키지를 부착합니다: 'InformationValue'
```

```
## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, sensitivity, specificity
```

```
library(ISLR)
K <- keras::backend()
```

Parameterization

```
# training parameters
vae_batch_size = 160L
fnn_batch_size = 160L
epochs = 1L
vae_ep = 1L
fnn_ep = 7L
vae_flag = 0L

sel_pr_up = 1.0 # upper bound probability for VAE
sel_pr_dw = 0.0 # lower bound probability for VAE
sel_rate = 1.0

vae_w1 = 0.0
vae_w2 = 0.0
vae_w3 = 1.0

# latent and intermediate dimension
latent_dim = 2L
intermediate_dim = 10L
epsilon_std <- 0.1

set.seed(50)
k=10

# input image dimensions
input_shape = c(k+1)
```

Data Generation

```
# 로지스틱 회귀모형

# Train 자료
# number of predictors and observations
nTR=80000
nTS=20000

# 회귀계수 생성
bet = c(1,1.5,2,2.5,0,-1,-1.5,-2,-2.5,0)

# 설명변수 생성, crash자료와 비슷하게 만들기 위하여 uniform에서 추출
xTR = matrix(runif(nTR*k), ncol=k)

# 반응변수 생성
library(LaplacesDemon)
yTR <- rbinom(nTR, 1, invlogit(xTR%*%bet))

dfTR = data.frame(y=yTR,x1=xTR[,1],x2=xTR[,2],x3=xTR[,3],x4=xTR[,4],x5=xTR[,5],x6=xTR[,6],x7=xTR[,7],x8=xTR[,8],x9=xTR[,9],x10=
xTR[,10])

# Test 자료

# 설명변수 생성, crash자료와 비슷하게 만들기 위하여 uniform에서 추출
xTS = matrix(runif(nTS*k), ncol=k)

# 반응변수 생성
library(LaplacesDemon)
yTS <- rbinom(nTS, 1, invlogit(xTS%*%bet))

dfTS = data.frame(y=yTS,x1=xTS[,1],x2=xTS[,2],x3=xTS[,3],x4=xTS[,4],x5=xTS[,5],x6=xTS[,6],x7=xTS[,7],x8=xTS[,8],x9=xTS[,9],x10=
xTS[,10])
```

반응변수가 1인 경우에 대한 downsampling

훈련자료에 대하여 반응변수가 1인 경우 downsampling을 하여 0에 비하여 비율이 매우 작은 자료를 생성

```
# down sampling

# downsampling probability
dpr = 0.0005 # keep dpr*100% only

dfTR0 = dfTR[dfTR[,1]==0,]
dfTR1 = dfTR[dfTR[,1]==1,]

downDF1 = dfTR1[sample(nrow(dfTR1), dpr*nrow(dfTR1)), ]

Downdf = rbind(dfTR0, downDF1)
Downdf = Downdf[sample(1:nrow(Downdf)),]
```

Oversampling with random copy

무작워로 반응변수가 1인 관측치들을 복제하여 적당한 수가 될 때까지 표본에 추가하는 방법을 적용

```
overDF1 = downDF1[sample(nrow(downDF1), nrow(dfTR1), replace=TRUE), ]

overdf = rbind(dfTR0, overDF1)
overdf = overdf[sample(1:nrow(overdf)),]
```

```
# encoder
original_input_size = c(k+1)
inp <- layer_input(shape = original_input_size)
x <- layer_lambda(inp, f=function(x) {x[,2:(k+1)]})
y <- layer_lambda(inp, f=function(x) {x[,1:1]})

hidden_1 <- layer_dense(x, units=intermediate_dim, activation="relu")
dropout_1 <- layer_dropout(hidden_1, rate = 0.5)
hidden_2 <- layer_dense(dropout_1, units=intermediate_dim, activation="relu")
dropout_2 <- layer_dropout(hidden_2, rate = 0.5)

z_mean = layer_dense(dropout_2, units = latent_dim)
z_log_var <- layer_dense(hidden_2, units = latent_dim)

# sampling part
sampling <- function(args) {
  z_mean <- args[, 1:(latent_dim)]
  z_log_var <- args[, (latent_dim + 1):(2 * latent_dim)]

  epsilon <- k_random_normal(
    shape = c(k_shape(z_mean)[[1]]),
    mean = 0.,
    stddev = epsilon_std
  )
  z_mean + k_exp(z_log_var) * epsilon
}

z <- layer_concatenate(list(z_mean, z_log_var)) %>% layer_lambda(sampling)

# decoder + prediction model
output_shape = c(vae_batch_size, k)

decoder_hidden = layer_dense(units=intermediate_dim, activation="relu")
decoder_upsample = layer_dense(units = intermediate_dim, activation="relu")
decoder_reshape <- layer_reshape(target_shape = intermediate_dim)
decoder_hidden1 = layer_dense(units=k, activation="sigmoid")

pred_layer = layer_dense(units = 1, activation = "sigmoid")
```

```
hidden_decoded  = decoder_hidden(z)
up_decoded = decoder_upsample(hidden_decoded)
reshape_decoded <- decoder_reshape(up_decoded)
hidden1_decoded = decoder_hidden1(reshape_decoded)

y_pred =pred_layer(hidden1_decoded)


vae_loss <- function(y, y_pred) {
  x <- k_flatten(x)
  x_decoded_mean_squash <- k_flatten(hidden1_decoded)
  xent_loss <- 1.0 * # initial weight = 1
    loss_mean_squared_error(x, x_decoded_mean_squash) # loss_categorical_crossentropy도 시도해 볼 것
  kl_loss <- -0.5 * k_mean(1 + z_log_var - k_square(z_mean) -  # initial weight = -0.5
                           k_exp(z_log_var), axis = -1L)
  p_loss <- 1.0 * loss_binary_crossentropy(y, y_pred) # initial weight = 0 * 12000

  k_mean(xent_loss*vae_w1 + kl_loss*vae_w2 + p_loss*vae_w3)
}

vae <- keras_model(inp, y_pred)
optimizers <- keras::keras$optimizers
vae %>% compile(optimizer = optimizers$legacy$RMSprop(learning_rate=0.0001), loss = vae_loss,
                metrics = c("accuracy"))
# summary(vae)

## encoder: model to project inputs on the latent space
# encoder <- keras_model(inp, list(z_mean, z_log_var))

## build a digit generator that can sample from the learned distribution
# gen_decoder_input <- layer_input(shape = latent_dim)
# gen_hidden_decoded <- decoder_hidden(gen_decoder_input)
# gen_up_decoded <- decoder_upsample(gen_hidden_decoded)
# gen_hidden1_decoded <- decoder_hidden1(gen_up_decoded)

# generator <- keras_model(gen_decoder_input, gen_hidden1_decoded)
```

```
vae1 <- keras_model(inp, hidden1_decoded) # can be used for generating synthetic samples for case 0 and 1

# summary(vae1)
```

model fitting

```
# j : number of epoch
# i : number of batchs for one epoch

for (j in 1:epochs) {

# FNN MODEL FITTING

model1 <- keras_model_sequential() %>%
  layer_dense(units = 1, activation = "sigmoid", input_shape = c(10))
  # %>%     layer_dense(units = 1, activation = "sigmoid")

model1 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

# Insert VAE part here if needed

if(vae_flag == 1){
history = vae %>% fit(
     as.matrix(overdf), as.matrix(overdf[,1]),
     shuffle = TRUE,
     epochs = vae_ep,
     batch_size = vae_batch_size,
     validation_data = list(as.matrix(dfTS), as.matrix(dfTS[,1])),
     verbose = 0
)
}


# whole train and test data preparation

 library(dplyr)

  temp0 <- predict(vae1, as.matrix(overdf))
  temp  <- predict(vae, as.matrix(overdf))
  temp1 <- as.data.frame(cbind(c(1), temp0[temp<=quantile(temp, sel_pr_up) &                          temp>=quantile(temp, s
```

```
el_pr_dw),]))
  names(temp1) = names(dfTR0)
  samp_ind = sample(1:nrow(temp), size = round(nrow(temp)*sel_rate))
  temp1 <- temp1[samp_ind,]

  temp2 <- dfTR0 %>% sample_frac(nrow(temp1)/nrow(dfTR0), replace = TRUE)
  train_df <-rbind(temp2, dfTR0, overDF1, temp1)
  train_df <- train_df[sample(1:nrow(train_df)),]

  # print(i)

  ## ---- Fitting -------------------------------------------------------------

  history2 <- model1 %>% fit(
    as.matrix(train_df[,-1]), as.matrix(train_df[,1]),
    shuffle = TRUE,
    epochs = fnn_ep, batch_size = fnn_batch_size,
    validation_data = list(as.matrix(dfTS[,-1]), as.matrix(dfTS[,1]))
    , verbose = 0
  )

  print("FNN")
  print(history2)
 # plot(history2)
  print("VAE")
  if(vae_flag == 1){
  print(history)}
 #plot(history)
  print(j)
 }
```

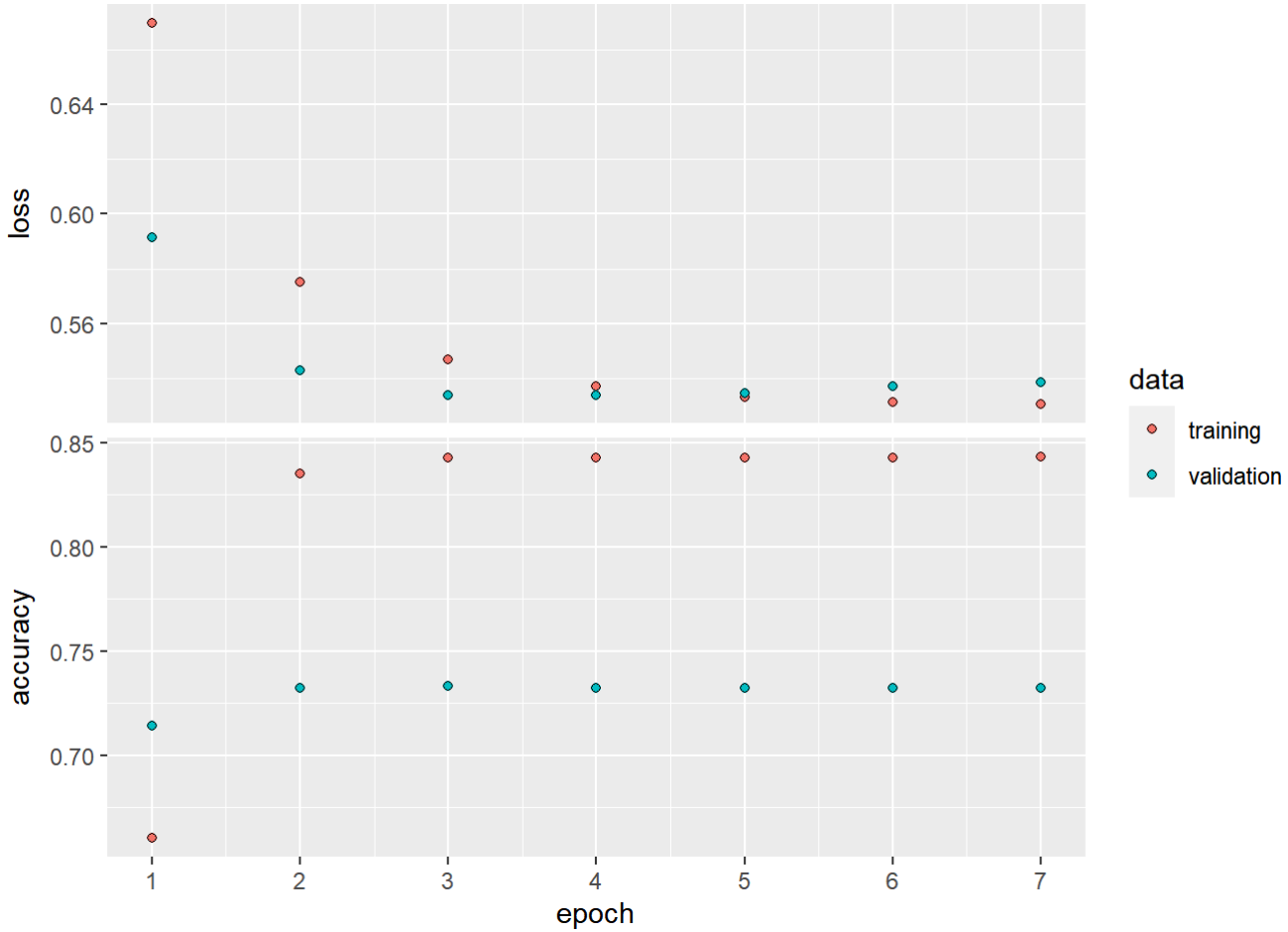
```
##
## 다음의 패키지를 부착합니다: 'dplyr'
```

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

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

```
## [1] "FNN"
## Trained on 240,000 samples (batch_size=160, epochs=7)
## Final epoch (plot to see history):
##         loss: 0.5304
##     accuracy: 0.8432
##     val_loss: 0.5385
## val_accuracy: 0.7321
## [1] "VAE"
## [1] 1
```

```
if(vae_flag == 1){
plot(history)
}
plot(history2)
```



test data accuracy

```
library(caret)
library(InformationValue)
library(ISLR)

temp3 <- predict(vae, as.matrix(dfTS))
confusionMatrix(temp3>=0.5, dfTS[,1]) -> tt
tt
```

	FALSE <int>
0	9996
1	10004

2 rows

total accuracy

```
(tt[1,1]+tt[2,2])/(sum(tt))*100
```

```
## numeric(0)
```

accuracy for case 0

```
tt[1,1]/(tt[1,1]+tt[1,2])*100
```

```
## numeric(0)
```

accuracy for case 1

```
tt[2,2]/(tt[2,1]+tt[2,2])*100
```

```
## numeric(0)
```

```
temp3 <- predict(model1, as.matrix(dfTS[,-1]))
confusionMatrix(temp3>=0.5, dfTS[,1]) -> tt3
tt3
```

	FALSE <int>	TRUE <int>
0	7101	2895

	FALSE <int>	TRUE <int>
1	2463	7541

2 rows

total accuracy

```
(tt3[1,1]+tt3[2,2])/(sum(tt3))*100
```

```
## [1] 73.21
```

accuracy for case 0

```
tt3[1,1]/(tt3[1,1]+tt3[1,2])*100
```

```
## [1] 71.03842
```

accuracy for case 1

```
tt3[2,2]/(tt3[2,1]+tt3[2,2])*100
```

```
## [1] 75.37985
```