STAT430_HW5

Taiga Hasegawa(taigah2)

2019/2/19

Practice 1

```
rm(list = setdiff(ls(), lsf.str()))
library(keras)
library(imputeTS)
library(Hmisc)
```

```
## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

## Loading required package: ggplot2

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```
dat <- read.csv("2017_M1_IEX.csv", header = F)
dat <- dat[,-3]
names(dat) <- c("tStamp", "ticker", "O", "H", "L", "C", "V")
```

```
## impute missing values by 0
dat$C <- zoo::na.locf(dat$C)
dat$V <- na.replace(dat$V, 0)
head(dat)
```

```
##                tStamp ticker      O       H      L       C    V
## 1 2017-01-03 09:31:00    SPY 224.94 224.940 224.84 224.850 2544
## 2 2017-01-03 09:32:00    SPY 224.85 224.895 224.85 224.890 1200
## 3 2017-01-03 09:33:00    SPY 224.86 224.995 224.86 224.995 1256
## 4 2017-01-03 09:34:00    SPY 225.01 225.140 225.01 225.130 1100
## 5 2017-01-03 09:35:00    SPY 225.18 225.240 225.18 225.210 2763
## 6 2017-01-03 09:36:00    SPY 225.18 225.250 225.14 225.140 1268
```

```
TICKER <- c("SPY", "AAPL", "MSFT", "AMZN", "BRK.B", "FB", "JNJ", "JPM", "XOM", "GOOG",
            "GOOGL", "BAC", "PFE", "UNH", "V", "T", "WFC", "CVX", "VZ", "HD", "INTC")
```

```r
## data to be included
col_included <- 1:(length(TICKER)*2)

allDat <- NULL
for(iTicker in TICKER)
{
  tmpDat <- subset(dat, ticker==iTicker)
  allDat <- cbind(allDat, tmpDat$C)
  allDat <- cbind(allDat, tmpDat$V)
}
```

```r
######################
# construct features #
######################

# number of trading days
nMin <- 60*6.5 # there are 30*13=390 minutes of each trading day
nDay <- dim(allDat)[1] / nMin

# 251 days in total
# train / val / test splt
train_days <- 1:150
val_days <- 151:200
test_days <- 201:nDay
train_min <- 1:(150*nMin)
val_min <- (150*nMin+1):(200*nMin)
test_min <- (200*nMin+1):(nDay*nMin)

length(train_min)
```

```
## [1] 58500
```

```r
length(val_min)
```

```
## [1] 19500
```

```r
length(test_min)
```

```
## [1] 19890
```

```r
X_data_train <- allDat[train_min, col_included]
X_data_val <- allDat[val_min, col_included]
X_data_test <- allDat[test_min, col_included]
```

Practice 2

```r
################################################
# use SPY next minute price direction as labels #
################################################
w <- 60
avgMprice <- c(rep(NA, w-1), zoo::rollmean(allDat[,1], k=w, align="left"))
preMP <- avgMprice
postMP <- c(avgMprice[-(1:w)], rep(NA,w))
length(preMP); length(postMP); dim(allDat)
```

```
## [1] 97890

## [1] 97890

## [1] 97890    42
```

```r
price_change <- postMP - preMP
r <- 0.08
Y <- rep(-1, length(preMP)) # stable
Y[price_change > r] <- 1 # increase
Y[price_change < - r] <- 0 # decrease
# early in the morning and late in the afternoon we don't have enough data to calculate the average
# here we exclude those labels as we don't want to use the features in the previous day to
# predict the labels in the next day
kept_min <- ( (1:nrow(allDat)) %% nMin >= w ) & ( (1:nrow(allDat)) %% nMin <= nMin - w )
Y[!kept_min] <- NA #
tb_Y_train <- table(Y[train_min])
tb_Y_val <- table(Y[val_min])
tb_Y_test <- table(Y[test_min])
rbind(tb_Y_train, tb_Y_val, tb_Y_test)
```

```
##               -1     0     1
## tb_Y_train 14459 10578 15613
## tb_Y_val    5559  3251  4740
## tb_Y_test   5181  3537  5103
```

```r
##############
# scale data #
##############

nCol <- ncol(X_data_train)
me_train <- apply(as.matrix(X_data_train), 2, mean)
sd_train <- apply(as.matrix(X_data_train), 2, sd)
me_val <- apply(as.matrix(X_data_val), 2, mean)
sd_val <- apply(as.matrix(X_data_val), 2, sd)

# rescale train data
for(i in 1:nCol) X_data_train[,i] <- scale(X_data_train[,i], center = me_train[i], scale = sd_train[i])
Y_data_train <- Y[train_min]

# rescale validation data (using train mean and sd)
for(i in 1:nCol) X_data_val[,i] <- scale(X_data_val[,i], center = me_train[i], scale = sd_train[i])
Y_data_val <- Y[val_min]

# rescale test data (using train mean and sd)
for(i in 1:nCol) X_data_test[,i] <- scale(X_data_test[,i], center = me_train[i], scale = sd_train[i])
Y_data_test <- Y[test_min]
```

Practice 3

```r
sampling_generator <- function(X_data, Y_data, batch_size, w)
{
  function()
  {
```

```
    rows_with_up_down <- w:nrow(X_data)

    Y <- X <- NULL
    Xlist <- list()
    size=0
    while(size!=batch_size){
      i <- sample( rows_with_up_down, 1, replace = TRUE )
      if(is.na(Y_data[i])){
        next
      }else{
        Xlist[[i]]=X_data[(i-w+1):i,]
        Y=c(Y,Y_data[i])
        size=size+1
      }
    }
    X <- array(abind::abind(Xlist, along = 0), c(batch_size, w, ncol(X_data), 1)) # add one axis of dim
    list(X, to_categorical(Y,num_classes = 3))
  }
}
```

```
w = 60
batch_size = 128
epochs = 5
```

```
library(keras)
use_condaenv("r-tensorflow")
k_clear_session()

model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 12, kernel_size = c(4, 20), activation = "relu", input_shape = c(60, 42, 1)) %
  layer_conv_2d(filters = 12, kernel_size = c(1, 1), activation = "relu") %>%
  layer_conv_2d(filters = 12, kernel_size = c(4, 1), activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 32, activation = "relu", kernel_regularizer = regularizer_l1(0.001)) %>%
  layer_dense(units = 3, activation = "sigmoid")

summary(model)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## conv2d (Conv2D)                  (None, 57, 23, 12)            972
## ___________________________________________________________________________
## conv2d_1 (Conv2D)                (None, 57, 23, 12)            156
## ___________________________________________________________________________
## conv2d_2 (Conv2D)                (None, 54, 23, 12)            588
## ___________________________________________________________________________
## flatten (Flatten)                (None, 14904)                 0
## ___________________________________________________________________________
## dropout (Dropout)                (None, 14904)                 0
## ___________________________________________________________________________
## dense (Dense)                    (None, 32)                    476960
## ___________________________________________________________________________
```

```
## dense_1 (Dense)                  (None, 3)                     99
## ===========================================================================
## Total params: 478,775
## Trainable params: 478,775
## Non-trainable params: 0
## ___________________________________________________________________________
```

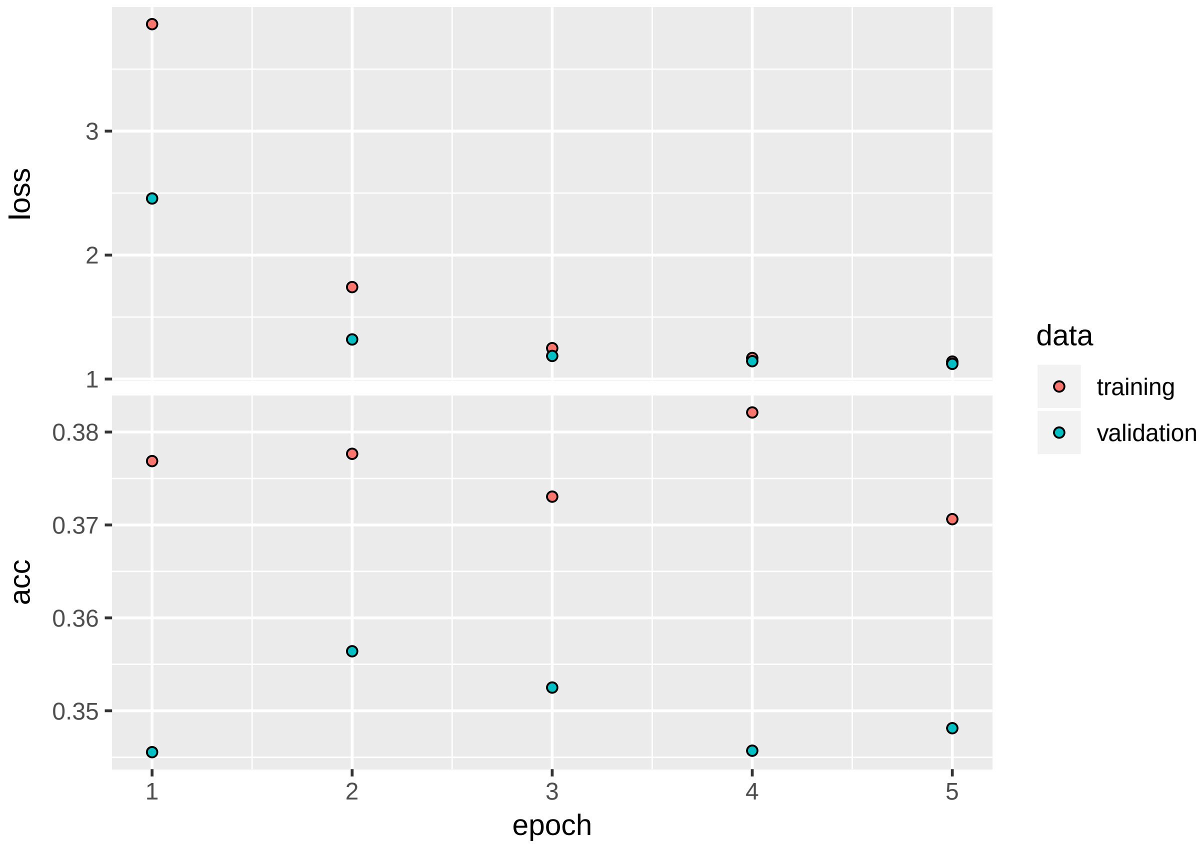
```r
model%>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(lr = 1e-4),
  metrics = c("accuracy")
)
```

```r
schedule <- function(epoch,lr) (lr)*(0.75^(floor(epoch/2)))
schedulLr <- callback_learning_rate_scheduler(schedule)

reduceLr <- callback_reduce_lr_on_plateau(monitor = "val_acc", factor = 0.1, patience = 3)

his <- model %>% fit_generator(sampling_generator(X_data_train, Y_data_train, batch_size = batch_size, v
                               steps_per_epoch = 100, epochs = epochs,
                               callbacks = list(reduceLr),
                               validation_data = sampling_generator(X_data_val, Y_data_val, batch_size =
                               validation_steps = 100)

plot(his)
```




```r
results <- model %>% evaluate_generator(sampling_generator(X_data_test, Y_data_test, batch_size = batch_
                                        steps = 100)
```

```r
results
```

```
## $loss
## [1] 1.130962
##
## $acc
## [1] 0.3714844
```