Homework #5

Larry Lei Hua

Apr 14, 2019

In total there are 30 pts:

(1) 5 pts for homework policy and presentation of the homework.
- If the files are not complete or not zipped into one file (should be one zipped file with one pdf and one Rmd), deduct 2 pts.
- If the overall presentation is messy and not organized, deduct 3 pts.
- If the data is also submitted, deduct 1 pt.

(2) 2 pts for Question 1
- The data should be correctly loaded and splitted.

(3) 5 pts for Question 2
- The critical places to check:
  - After the features and labels are prepared, the following are the distributions among different sets. The numbers should be very close but not need to be exactly the same. Deduct 4 pts if the numbers are very different.
    * the numbers of labels for 0, 1, 2 (decrease, stable, increase) are:
    * 10565, 14398, 15687 for the training set;
    * 3257, 5563, 4730 for the validation set;
    * 3545, 5173, 5103 for the test set.

(4) 3 pts for Question 3
- The data generator should be correctly created.

(5) 11 pts for Question 4
- The critical places to check:
  - 2 pts for missing values. The missing values should be taken care of; if not, deduct 2 pts. If the missing values of volumes are imputed by a non-zero number, then deduct 1 pts. (We shouldn't add volumes if they are none.)
  - 2 pts for scaling the features. If the features are not scaled based on the training set, deduct 2 pts.
  - 3 pts for correctly using callback functions. If no callback functions are used, deduct 3 pts.
  - 3 pts for constructing CNN models.
  - 1 pts for displaying the history plot that contains at least acc and val_acc.

(6) 4 pts for Question 5
- The critical places to check:
  - 2 pts for using the correct model and data to evaluate. The `best` model should be used to evaluate the model based on the test set, and the best model should be loaded from a saved file `HW5.h5` which was in a callback function. If not, deduct 2 pts.

– 2 pts for meeting the requirement of performance.

(Please read the Homework Policy before you start)

Description: In this homework, you will practice using convolutional neural network to account for the cross-sectional and short-term temporal dependence among multiple time series. Meanwhile, you will practice how to construct features, labels, and the data generator for handling relatively larger dataset for a particular task.

Objective: Predict average price movement direction in the next minute for SPY (SP500 ETF), based on the close prices and volumes in the previous hour (60 minutes) of SPY and 20 other major stocks provided.

Dataset: The same dataset used for Homework #4. This is one-minute time bar data of SPY and 20 largest component stocks of SPY. The data has been pre-processed from the tick data provided by IEX, and has been provided as is and for your homework only.

Practices:

1. There are 251 trading days in total from the dataset. Choose 1 to 150 as training set, 151 to 200 as validation set, and 201 to 251 as test set.

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

work_folder <- "~/Dropbox/Teaching/STAT430/homework/hw05"
dat <- read.csv("~/Dropbox/Teaching/STAT430/datasets/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)

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

## 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)
  }

  ######################
  # 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
```

```
  length(val_min)
```

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

```
  length(test_min)
```

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

2. Prepare features and labels:
   - The first observation on each day takes all the close prices and volumes of all the 21 tickers during minutes 1~60 as features, and the price movement direction (0,1,2) on minute 61 as the label
   - The price movement direction on minute 61 is calculated based on the comparison between the following two values and a threshold $r = 0.08$, similar to the LOB example discussed in class:
     - average close price for minutes 1~60 vs average close price for minutes 61~120
   - The features and labels are created each day according to the above procedure. Therefore, for each day there are 271 observations with labels taking the price movements directions on minutes 61~331, respectively.
   - [Hint 1] Be careful about missing values of prices and volumes, and we should use appropriate methods to impute the missing values,

respectively.

- [Hint 2] After creating the labels, the following are distributions of the labels. Before moving on to the next step, you should make sure that your obtained numbers are roughly the same as follows:
  - the numbers of labels for 0, 1, 2 (decrease, stable, increase) are:
    * 10565, 14398, 15687 for the training set;
    * 3257, 5563, 4730 for the validation set;
    * 3545, 5173, 5103 for the test set.

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

################################################
# 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
```

```
price_change <- postMP - preMP

r <- 0.08
Y <- rep(1, length(preMP)) # stable
Y[price_change > r] <- 2 # 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])
```

```r
  tb_Y_val <- table(Y[val_min])
  tb_Y_test <- table(Y[test_min])
  rbind(tb_Y_train, tb_Y_val, tb_Y_test)
```

```
##                0     1     2
## tb_Y_train 10578 14459 15613
## tb_Y_val    3251  5559  4740
## tb_Y_test   3537  5181  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]
```

3. Create appropriate data generator so that a large dataset like this one can be fed into your models.

```r
  ##########################
  # create data generator #
  ##########################
  sampling_generator <- function(X_data, Y_data, batch_size,w=60)
  {
    function()
    {
```

```
    # we sample based on available Y only
    rows <- sample(which(!is.na(Y_data)), batch_size, replace = TRUE)
    tmp <- Y <- X <- NULL

    # for X
    for(i in rows) tmp <- rbind(tmp, as.vector(as.matrix(X_data[((i-w+1):i),])))
    X <- array_reshape(tmp, c(batch_size, w, ncol(X_data), 1), order = "F")

    # for Y
    Y <- to_categorical(Y_data[rows], num_classes = 3)
    list(X, Y)
  }
}
```

4. Construct convolutional neural network appropriately to conduct classification of these 3 classes; do not use recurrent neural network for this homework, but you can try any techniques you have learned so far to increase the performance. The following components should be included in your analysis:

```
#############
# callbacks #
#############

# we have to use the call back function to store the best model that is to be used for evaludation of test set
checkPoint <- callback_model_checkpoint(filepath = file.path(work_folder, "HW5.h5"),
                                        monitor = "val_acc", save_best_only = TRUE)

reduceLr <- callback_reduce_lr_on_plateau(monitor = "val_acc", factor = 0.1, patience = 3)
logger <- callback_csv_logger(file.path(work_folder, "logger.csv"))

#################
# build convnet #
#################
k_clear_session()
model <- keras_model_sequential() %>%
    layer_conv_2d(filters = 6, kernel_size = c(3, 3), activation = "relu", input_shape = c(w, nCol, 1)) %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_conv_2d(filters = 8, kernel_size = c(3, 3), activation = "relu") %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_flatten() %>%
```

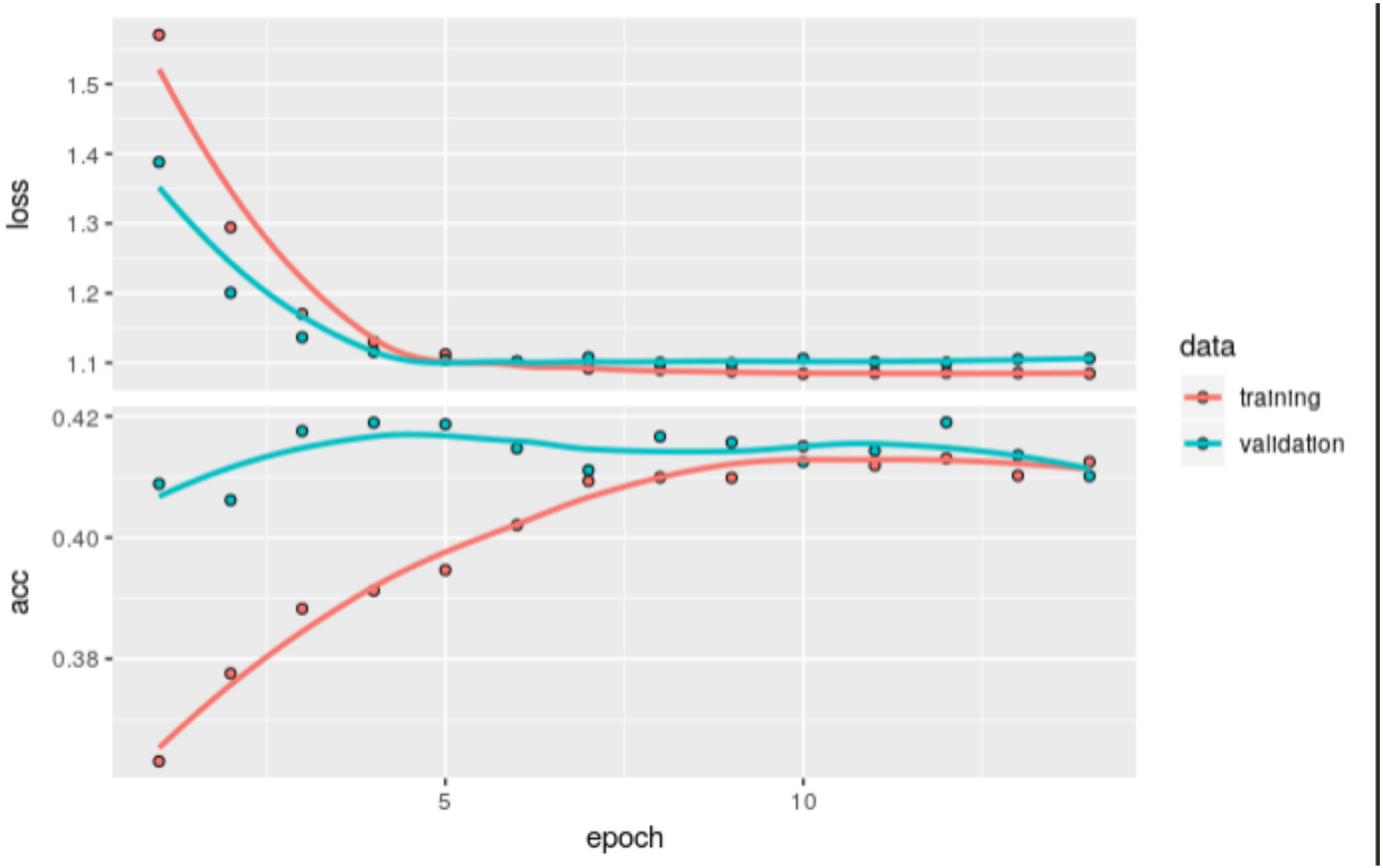
```
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 16, activation = "relu", kernel_regularizer = regularizer_l1(0.001)) %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 3, activation = "softmax")

model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(lr = 1e-4),
  metrics = c("acc")
)
batch_size <- 120
```

5. Try different techniques you have learned so far to tweak the model so that the following criteria are met:
   - accuracy based on the validation set $> 47\%$
   - accuracy based on the test set $> 39\%$
   - If you cannot achieve the above criteria, please use the `acc_lucky()` function discussed in class to show that the accuracy based on your model is better than the 3 types of guesses.
     - A blog post about classification accuracy

```
#################
# run the model #
#################
# This part takes time, so I only run it before kniting the Rmarkdown,
# and then save the results and the plots for later use.
his <- model %>% fit_generator(sampling_generator(X_data_train, Y_data_train, batch_size = batch_size, w=w), epochs = 14,
                               steps_per_epoch = floor( length(Y_data_train) / batch_size),
                               callbacks = list(checkPoint, reduceLr, logger),
                               validation_data = sampling_generator(X_data_val, Y_data_val, batch_size = batch_size, w=w),
                               validation_steps = floor( length(Y_data_train) / batch_size)
)
plot(his)
```

```
knitr::include_graphics("/home/lh/Dropbox/Teaching/STAT430/homework/hw05/hw5-his.png")
```




```
fitted <- load_model_hdf5(file.path(work_folder, "HW5.h5"))
results <- fitted %>%
  evaluate_generator(
    sampling_generator(X_data_test, Y_data_test, batch_size =batch_size, w=w),
    steps = floor(length(Y_data_test)/batch_size))
```

```
  results
```

```
## $loss
## [1] 1.129256
##
## $acc
## [1] 0.384596
```

```
  fmlr::acc_lucky(as.vector(tb_Y_train), as.vector(tb_Y_test), results$acc)
```

```
## $my_accuracy
## [1] 0.384596
##
## $p_random_guess
## [1] 0
##
## $p_educated_guess
## [1] 0
##
## $mean_random_guess
## [1] 0.3333517
##
## $mean_educated_guess
## [1] 0.3417753
##
## $acc_majority_guess
## [1] 0.3692208
```