1 Cross Validation

Script `00-1-kfold.R` performs k-fold cross validations for n variations of settings for a neural network. In order to perform a k-fold cross validation, the dataset is randomly split into k subsets folds(line 5). For every run i of the k-fold cross validation subset `folds==i` is left out for prediction, which means that the model is trained on the subset without `folds==i` and the prediction is performed on subset `fold==i`. This split is performed in lines 17-22, followed by training the model (line 24) and prediction (line 29). Note that the second @@@@laufindex n indicates the settings for the neural network (line 27+28). This index n is changed by the outer foreach-loop (line 11).
To evaluate the performance of the model, the loss **??** is used as a measure (line 30).

In order to decrease run time, the script uses the package `"doParallel"` for parallel computing. Lines 7-9 define the settings for parallel computing. In line 14 the nesting operator **%dopar%** starts parallel computing, so that different runs of the foreach-loop can be performed by a different cores at the same time.
The output of the k-fold cross validation is a k*n matrix (k: k-fold cross validation, n: number of variations of settings for model), where every element consists of a list of the measure and the settings of the model (line 36).

```
1  # 00-1-kfold_cv.R
2  k = k               # dimension of cv is choosen in parent script
3  sample.idx = sample(nrow(tr.v))
4  train.rnd  = tr.v[sample.idx,]
5  folds      = cut(1:nrow(train.rnd), breaks = k, labels = FALSE)
6  # settings for parallel computing
7  nrOfCores = detectCores()
8  cl        = makeCluster( max(1,detectCores()-1))
9  registerDoParallel(cl)
10 # loops for cross validation and tuning
11 results.par = foreach(n = 1:nrow(parameters), .combine = cbind,
12                       .packages = c("caret", "nnet", "data.table"
                           )) %:%
13     foreach(i = 1:k,
14             .packages = c("caret","nnet", "data.table")) %dopar%{
15         set.seed(1234) # set seed for reproducibility
16         # Split data into training and validation
17         idx.val   = which(folds == i, arr.ind = TRUE)
18         cv.train  = train.rnd[-idx.val,]
19         cv.train  = cv.train[order(cv.train$order_item_id),]
20         cv.val    = train.rnd[idx.val,]
21         cv.val    = cv.val[order(cv.val$order_item_id),]
22         price     = real_price$item_price[cv.val$order_item_id]
23         # train nnet and make prediction
24         nnet      = nnet(return~. -order_item_id - tau,
25                          data  = cv.train,
26                          trace = FALSE, maxit = 1000,
27                          size  = parameters$size[n],
28                          decay = parameters$decay[n])
29         yhat.val  = predict(nnet, newdata = cv.val, type = "raw")
30         loss      = helper.loss(tau_candidates = tau_candidates,
31                                 truevals       = cv.val$return,
32                                 predictedvals  = yhat.val,
33                                 itemprice      = price)
34         pars      = data.table("size"  = parameters$size[n],
```

```
35                                  "decay" = parameters$decay[n])
36          res       = list("loss"         = max(loss),
37                           "parameters"   = pars)
38          }
39  stopCluster(cl) # stop parallel computing
```

Since the performance of the model also depends on the "quality" of the training data and the training data is determined randomly, it is crucial to change the randomness of the splits. By varying the seed `set.seed(1234*t)` in the mother script `00-2-rep_cv.R` , the randomness of the training set tr is changed (line 11-16).
Further this script `00-2-rep_cv.R` splits the dataset according to the six tau-categories **??**. The k-fold cross validation from the previous section is then performed for every of these subsets. The results for each tau-category `v` are then stored in a list `tau_c` (line 22).
The output of the m-times repeated cross validation is a list of m times the same cross validation procedure (only with different randomness). Each element of the m-sized list is a list of the k*n matrix for the six tau-categories v.
(m : number of repeated cross validations (same procedure but different randomness), v : six tau-categories, k : k-fold cross validation, n : number of variations of settings for model)

```
1  tau_c    = list()
2  cv.list = list()
3
4  for (t in 1:m){
5      set.seed(1234*t)
6      # Splitting the data into a test and a training set
7      idx.train = caret::createDataPartition(y = known$return, p =
         0.8, list = FALSE)
8      # Actual data splitting
9      tr = known[idx.train,  ] # training set
10     ts = known[-idx.train, ] # test set
11     for (v in 1:6){
12         # call script for each tau-class
13         tr.v = tr[tr$tau == v, ]
14         print(paste("tau =", v, "rep = ", t))
15         source(file = "./nnet/00-1-kfold_cv.R")
16         tau_c[[paste("tau_c ==", v)]] = results.par
17   }
18   cv.list[[t]] = tau_c
19 }
```

Section **??** explains the importance of decomposing our dataset into four subsets according to the "quality" @@@aussagekraft of the features for training. In script `01-par-tuning.R` the whole tuning-process is performed for the different subsets. The following explains the preparation of the .u subset exemplarily.
The known dataset consists of the subsets where a full model can be trained (.f) and the subset where the model without `user_retrate` can be trained (`.u`). These two subsets are combined to the known data set (line 51). For training a neural network additional data preparation as described in section **??** is necessary (line 54). Then the columns of the feature user_retrate get removed and the set is named according to the requirements of the script for cross validation (lines 55-59).

After performing the m times repeated k-fold cross validation for the n variations of settings for the model, the results are stored in a the list `cv.list.u` for further evaluation (line 71).

```
known    = rbind(known.f, known.u) # can also use .f for training,
   variables are pure
unknown = unknown.u                #
# additional data preparation for nnet
source(file = "./nnet/00-3-nnet_DataPrep.R")
known.n$user_retrate    = NULL     # remove user_retrate (
  uncertain categories)
unknown.n$user_retrate = NULL      #
                                   #
known    = known.n                 # output of additional data
  preparation
unknown = unknown.n                #

# perform repeatet cross validation
source(file = "./nnet/00-2-rep_cv.R")
cv.list.u    = cv.list             # store result of repeated
  cross validation
```