Deep Learning in R - Glötzl Alexander

Prediction

I tried several times to get my GPU (Geforce GTX970) running with Keras but I couldn't quite manage to do so. Therefore for the project I focused on the Random Forest and XGBoost models since they are not as computational heavy to train.

Random Forest for the Wine and Titanic data set

For both data sets I used a 5-fold cross-validation to not overtrain my model on the training data.

For the Titanic data set Random Forest with a grid search for the hyperparameters `nodesize` and `maxnodes` was used. Since we only considered four features in the Titanic data set in total I kept `mtry` equal to 2. The model should predict wether passengers survived or not.

For the Wine data set `mtry` was also included in the hyperparameters. Also the performance score had to be changed from AUC to accuracy since AUC does not support multiple classes. The task of the model was to predict the correct quality of the wine (classification).

XGBoost for the NASA data set

Again for the XGBoost algorithm a grid search was applied, varying the `depth` and `eta` parameters. The aim is to predict whether the asteroids are hazardous or not.

Convolutional Neural Network for the flower data set

I used three convolutional layers each followed by a max-pooling- and dropout-layer. The convolutional layers were then flattened and used as input for four additional dense layers. The goal was to predict the flower type. The CNN performed poorly for me because I could not get the model complex enough for it to overfit. Therefore it performed bad even on the training set. I tried to add as much layers as I could and also disabled any regularization. The chosen optimizer for the loss function was `adamax` since it converges slower but more precise than `sgd`.

Code

Random Forest - Titanic data set

```
#eval=FALSE
library(randomForest)
library(EcoData)
library(dplyr)
library(missRanger)
library(tidyverse)

# TITANIC data set
data(titanic_ml)
data = titanic_ml
data =
```

```
  data %>% select(survived, sex, age, fare, pclass)

data[,-1] = missRanger(data[,-1], verbose = 0)

data_sub =
  data %>%
    mutate(age = scales::rescale(age, c(0, 1)),
           fare = scales::rescale(fare, c(0, 1))) %>%
    mutate(sex = as.integer(sex) - 1L,
           pclass = as.integer(pclass - 1L))

data_new = data_sub[is.na(data_sub$survived),] # for which we want to make predictions at the end
data_obs = data_sub[!is.na(data_sub$survived),] # data with known response

cv <- 5
outer_split = as.integer(cut(1:nrow(data_obs), breaks = cv))

#Grid search
my_nodesize = c(1,2,3,4,5,6,7)
my_maxnodes=c(30,35,37,40,42,45,50)

myGrid <- expand.grid(nodesize = my_nodesize, maxnodes=my_maxnodes)

results = data.frame(
  set = rep(NA, nrow(myGrid)),
  AUC = rep(NA, nrow(myGrid)),
  nodesize = rep(NA, nrow(myGrid)),
  maxnodes = rep(NA, nrow(myGrid))
)

# Five-Fold Cross-validation
for (hyperp in 1:nrow(myGrid)){
    cv_tuning_results = data.frame(
                        set = rep(NA, cv),
                        AUC = rep(NA, cv),
                        nodesize = rep(NA, cv),
                        maxnodes = rep(NA, cv))
    for(i in 1:cv) {
        train_outer = data_obs[outer_split != i, ]
        test_outer = data_obs[outer_split == i, ]

        train_outer$survived <- as.factor(train_outer$survived)
        test_outer$survived <- as.factor(test_outer$survived)
        model <- randomForest(survived~., data = train_outer, mtry=2, nodesize=myGrid[hyperp,1], maxnode

        tuning_results = Metrics::auc(test_outer$survived, predict(model, test_outer[,-c(1)], type="prob

        cv_tuning_results[i,] <- c(i, tuning_results, myGrid[hyperp,1], myGrid[hyperp,2])
    }
    tuning_results_avg <- mean(cv_tuning_results[,2])

    results[hyperp, 1] = hyperp
    results[hyperp, 2] = tuning_results_avg
```

```
    results[hyperp, 3] = myGrid[hyperp,1]
    results[hyperp, 4] = myGrid[hyperp,2]
}
print(head(results))
```

```
##   set       AUC nodesize maxnodes
## 1   1 0.8393343        1       30
## 2   2 0.8357779        2       30
## 3   3 0.8355333        3       30
## 4   4 0.8379704        4       30
## 5   5 0.8380508        5       30
## 6   6 0.8351616        6       30
```

```
best_hyperparameters <- results[min(which(results$AUC == max(results$AUC))),]
best_hyperparameters
```

```
##    set       AUC nodesize maxnodes
## 31  31 0.8403179        3       42
```

```
data_obs$survived <- as.factor(data_obs$survived)
model <- randomForest(survived~., data = data_obs, mtry=2, nodesize=best_hyperparameters$nodesize, maxn

data_new = data_sub[is.na(data_sub$survived),]
#write.csv(data.frame(y = predict(model, data_new[,-c(1)], type="prob")[,2] ), file = "../predictions/r
```

Random Forest - Wine data set

```
#WINE data set
data(wine)

wine_imputed = missRanger::missRanger(data = wine %>% select(-quality), verbose = 0)
wine_imputed$quality = wine$quality

wine_imputed[,-c(12)] <- scale(wine_imputed[,-c(12)])

train = wine_imputed[!is.na(wine$quality), ]
test = wine_imputed[is.na(wine$quality), ]

#Random Forest
my_nodesize = c(2,3,4,5)            #2    #3    #3
my_maxnodes=c(60,61,62,63,64)       #50   #60   #62
my_mtry = c(6,7,8,9,10)             #8    #11   #8

myGrid <- expand.grid(nodesize = my_nodesize, maxnodes=my_maxnodes, mtry = my_mtry)
cv <- 5

results = data.frame(
  set = rep(NA, nrow(myGrid)),
  AUC = rep(NA, nrow(myGrid)),
  nodesize = rep(NA, nrow(myGrid)),
  maxnodes = rep(NA, nrow(myGrid)),
  mtry = rep(NA, nrow(myGrid))
)
```

```
outer_split = as.integer(cut(1:nrow(train), breaks = cv))

for (hyperp in 1:nrow(myGrid)){
    cv_tuning_results = data.frame(
                        set = rep(NA, cv),
                        AUC = rep(NA, cv),
                        nodesize = rep(NA, cv),
                        maxnodes = rep(NA, cv),
                        mtry =     rep(NA, cv))
    for(i in 1:cv) {
        train_outer = train[outer_split != i, ]
        test_outer = train[outer_split == i, ]

        train_outer$quality <- as.factor(train_outer$quality)
        test_outer$quality <- as.factor(test_outer$quality)

        model <- randomForest(quality~., data = train_outer, mtry=myGrid[hyperp,3], nodesize=myGrid[hyp

        my_pred <- predict(model, test_outer[,-c(dim(test_outer)[2])], type="class")
        my_pred_non_factor <- as.numeric(levels(my_pred))[my_pred]
        tuning_results = mean(as.numeric(levels(test_outer[,c(12)]))[test_outer[,c(12)]] == my_pred_non

        cv_tuning_results[i,] <- c(i, tuning_results, myGrid[hyperp,1], myGrid[hyperp,2], myGrid[hyperp
    }
    tuning_results_avg <- mean(cv_tuning_results[,2])

    results[hyperp, 1] = hyperp
    results[hyperp, 2] = tuning_results_avg
    results[hyperp, 3] = myGrid[hyperp,1]
    results[hyperp, 4] = myGrid[hyperp,2]
    results[hyperp, 5] = myGrid[hyperp,3]
}
best_hyperparameters <- results[which(results$AUC == max(results$AUC)),]
best_hyperparameters
```

```
##    set      AUC nodesize maxnodes mtry
## 78  78 0.613815        3       64    9
## 82  82 0.613815        3       60   10
```

```
train$quality <- as.factor(train$quality)
model <- randomForest(quality~., data = train, mtry=best_hyperparameters$mtry, nodesize=best_hyperparam
```

```
## Warning in if (maxnodes > nrnodes) warning("maxnodes exceeds its max value."):
## the condition has length > 1 and only the first element will be used
```

```
#write.csv(data.frame(y = predict(model, test[,-c(12)]) ), file = "../predictions/rforest_grid_alex_lin
```

XGBoost - NASA data set

```
#NASA DATA with grid search
library(xgboost)
data(nasa)

nasa = nasa[, unlist(lapply(nasa, is.numeric))]
```

```
nasa_imputed = missRanger::missRanger(data = nasa %>% select(-Hazardous),
                                      maxiter = 1, num.trees = 5L, verbose = 0)
nasa_imputed$Hazardous = nasa$Hazardous

data_obs = nasa_imputed[!is.na(nasa$Hazardous), ] #train
data_new = nasa_imputed[is.na(nasa$Hazardous), ] #test

cv = 3
outer_split = as.integer(cut(1:nrow(data_obs), breaks = cv))

hyper_depth = c(6, 112, 131)       #sample(200, 20)
hyper_eta = seq(0,1,0.01) #runif(20, 0, 1)

myGrid <- expand.grid(hyper_depth = hyper_depth, hyper_eta=hyper_eta)

results = data.frame(
  set = rep(NA, nrow(myGrid)),
  AUC = rep(NA, nrow(myGrid)),
  depth = rep(NA, nrow(myGrid)),
  eta = rep(NA, nrow(myGrid))
)

for (hyperp in 1:nrow(myGrid)){
    cv_tuning_results = data.frame(
                        set = rep(NA, cv),
                        AUC = rep(NA, cv),
                        depth = rep(NA, cv),
                        eta = rep(NA, cv))

    for(i in 1:cv) {
        train_outer = data_obs[outer_split != i, ]
        test_outer = data_obs[outer_split == i, ]

        data_xg = xgb.DMatrix(data = as.matrix(train_outer[,-c(ncol(train_outer))]), label = train_outer
        model = xgboost(data_xg, nrounds = 16L, eta = myGrid[hyperp,"hyper_eta"], max_depth = myGrid[hyp

        predictions = predict(model, newdata = as.matrix(test_outer[,-c(ncol(test_outer))]))
        tuning_results = Metrics::auc(test_outer$Hazardous, predictions)

        cv_tuning_results[i, 1] = i
        cv_tuning_results[i, 2] = max(tuning_results)
        cv_tuning_results[i, 3] = hyper_depth[which.max(tuning_results)]
        cv_tuning_results[i, 4] = hyper_eta[which.max(tuning_results)]

    }

    tuning_results_avg <- mean(cv_tuning_results[,2])

    results[hyperp, 1] = hyperp
    results[hyperp, 2] = tuning_results_avg
    results[hyperp, 3] = myGrid[hyperp,1]
    results[hyperp, 4] = myGrid[hyperp,2]
}
```

```
best_hyperparameters <- results[min(which(results$AUC == max(results$AUC))),]
best_hyperparameters
```

```
##     set       AUC depth  eta
## 277 277 0.9703964     6 0.92
```

```
data_xg = xgb.DMatrix(data = as.matrix(data_obs[,-c(ncol(data_obs))]), label = data_obs$Hazardous)

model = xgboost(data_xg, nrounds = 16L, eta = best_hyperparameters$eta, max_depth = best_hyperparameters

data_test_xg = as.matrix(data_new[,-c(ncol(data_new))])
#write.csv(data.frame(y = predict(model, newdata=data_test_xg)), file = "../predictions/xgb_grid_aleX.c
```

Convolutional NN - Flower data set

```
library(tensorflow)
library(keras)

load(file='../data/flower.rda')

train = flower$train/255
test = flower$test/255
labels = flower$labels

model = keras_model_sequential()
model %>%
  layer_conv_2d(filters = 80L, kernel_size = 4L,
                input_shape = list(80L, 80L, 3L),
                #kernel_regularizer = regularizer_l1(0.03),
                activation = "relu") %>%
  layer_max_pooling_2d() %>%
  layer_dropout(0.25) %>%
  layer_conv_2d(filters = 50L, kernel_size = 4L,
                #kernel_regularizer = regularizer_l1(0.03),
                activation = "relu") %>%
  layer_max_pooling_2d() %>%
  layer_dropout(0.25) %>%
  layer_conv_2d(filters = 30L, kernel_size = 4L,
                #kernel_regularizer = regularizer_l1(0.03),
                activation = "relu") %>%
  layer_max_pooling_2d() %>%
  layer_dropout(0.25) %>%
  layer_flatten() %>%
  layer_dense(1000L, activation = "relu", kernel_regularizer = regularizer_l1(0.01)) %>%
  layer_dense(500L, activation = "relu", kernel_regularizer = regularizer_l1(0.01)) %>%
  layer_dense(100L, activation = "relu", kernel_regularizer = regularizer_l1(0.01)) %>%
  layer_dense(30L, activation = "relu", kernel_regularizer = regularizer_l1(0.01)) %>%
  layer_dense(units = 5L, activation = "softmax")


### Model fitting ###
model %>%
  compile(loss = loss_categorical_crossentropy,
```

```
          optimizer = optimizer_adamax(learning_rate = 0.01))

model_history <- model %>%
  fit(x = train, y = keras::k_one_hot(labels, 5L), epochs = 10L, batch_size = 100L)

plot(model_history)

# Prediction on training data:
pred_train = apply(model %>% predict(train), 1, which.max)
Metrics::accuracy(pred_train - 1L, labels) # only 0.2434668 on the training set!

#test data:
pred_test = model %>% predict(test) %>% apply(1, which.max) - 1L
table(pred_test)

mean((pred_train - 1L) == labels)
#write.csv(data.frame(y = pred_test), file = "../predictions/CNN_roy_daniel_alex.csv")
```

Results

Test accuracy on the Titanic, Wine, NASA and flower data set:

	Titanic	Wine	NASA	Flower
Random Forest	0.878121	0.622099		
XGBoost			0.996480	
Conv. NN				0.632308