Grid Search Parameter Optimization

Lucas Jamar

2020-05-31

Using a lightGBM model, a grouped 6-fold cross-validation, and various model parameters, we train multiple L2 regression models of the difference between the true and expected portion of remaining HP 1. Like with our validation set, we use Name_2 to group each opponent into the same fold. Afterwards, Name_2 is removed from the list of features. The performance of the models with different parameters is evaluated based on the RMSE, MAE and R2 of the validation predictions. The results of each model are appended to a CSV. This CSV is later used to determine the optimal parameters for training the model on the entire data.

Read in the data with features

```
library(MLmetrics)
library(caret)
library(lightgbm)
library(data.table)

train_dt <- data.table::fread("data/04_features/pokemon.csv")
```

Features to keep for modelling

```
feature_columns <- c(
  "Name_1",
  "Level_1",
  "Price_1",
  "HP_1",
  "Attack_1",
  "Defense_1",
  "Sp_Atk_1",
  "Sp_Def_1",
  "Speed_1",
  "Legendary_1",
  "Level_2",
  "Price_2",
  "HP_2",
  "Attack_2",
  "Defense_2",
  "Sp_Atk_2",
  "Sp_Def_2",
  "Speed_2",
  "Legendary_2",
  "WeatherAndTime",
  "MaxStrength_1",
  "MaxStrength_2",
  "Type_1",
  "Type_2",
  "WeatherInfluence_1",
  "WeatherInfluence_2",
  "Modifier_1",
  "Modifier_2",
  "Damage_1",
  "Damage_2",
  "Sp_Damage_1",
  "Sp_Damage_2",
  "MaxDamage_1",
  "MaxDamage_2",
  "ExpectedRounds_1_ToDefeat_2",
  "ExpectedRounds_2_ToDefeat_1",
  "ExpectedRounds",
  "ExpectedRemainingHP_1",
  "ExpectedRemainingHP_2",
  "RatioLevel",
  "RatioPrice",
  "RatioHP",
  "RatioAttack",
  "RatioDefense",
  "RatioSp_Atk",
  "RatioSp_Def",
  "RatioSpeed",
  "RatioWeatherInfluence",
  "RatioDamage",
  "RatioSp_Damage",
  "RatioMaxDamage",
  "RatioExpectedRounds_ToDefeat",
  "RatioExpectedRemainingHP",
  "RatioMaxStrength"
)
```

Separate train from test data

```
test_dt <- train_dt[Set =="test"]
train_dt <- train_dt[Set =="train"]
```

Separate target from features

```
test_result <- test_dt[, .(Set, PortionRemainingHP_1, ExpectedPortionRemainingHP_1, TrueMinusExpectedPortion
RemainingHP_1)]
train_result <- train_dt[, .(Set, PortionRemainingHP_1, ExpectedPortionRemainingHP_1, TrueMinusExpectedPorti
onRemainingHP_1)]
```

Create grouped K-Fold indices around opponent Name. Caret provides the indices of train data per fold but we need the indices of validation data per fold.

```
set.seed(12345)
grouped_folds = caret::groupKFold(group = train_dt$Name_2, k = 6)
indices = 1:nrow(train_dt)
for(fold in 1:length(grouped_folds)) {
  grouped_folds[[fold]] = indices[!indices %in% grouped_folds[[fold]]]
}
```

Keep only features for model training

```
keep_columns <- function(df, columns) {
  columns <- colnames(df)[!colnames(df) %in% columns]
  df[, (columns) := NULL]
}
keep_columns(train_dt, feature_columns)
keep_columns(test_dt, feature_columns)
```

Determine which features are categorical

```
features <- colnames(train_dt)
categorical_features <- features[lapply(train_dt, class) == "character"]
```

Encode categorical variables using LGB encoder

```
train_dt <- lgb.prepare_rules(data = train_dt)
rules <- train_dt$rules
test_dt <- lgb.prepare_rules(data = test_dt, rules = rules)
train_dt <- as.matrix(train_dt$data)
test_dt <- as.matrix(test_dt$data)
```

Convert to LGB datasets

```
dtrain <- lgb.Dataset(
  label = train_result$TrueMinusExpectedPortionRemainingHP_1,
  data = train_dt,
  categorical_feature = categorical_features,
  free_raw_data = FALSE
)

dtest <- lgb.Dataset(
  label = test_result$TrueMinusExpectedPortionRemainingHP_1,
  data = test_dt,
  categorical_feature = categorical_features,
  free_raw_data = FALSE
)

valids <- list(train = dtrain, test = dtest)
```

Create grid of parameters to explore

```
parameters <- list(
  nthread = -1,
  boosting = c("goss"),
  num_iterations = 3 * 10^4,
  learning_rate = c(0.1),
  feature_fraction = c(0.95),
  num_leaves = c(30),
  seed = 12345
)
parameters <- data.table::data.table(expand.grid(parameters))
results <- data.table::copy(parameters)
```

Grid search with 6-fold CV

```r
for (row in 1:nrow(parameters)) {
  parameter <- parameters[row]
  results[row, time_start := Sys.time()]

  # Perform 6-fold CV to determine optimal number of boosting iterations
  lgb_pokemon <- lgb.cv(
    data = dtrain,
    objective = "regression",
    folds = grouped_folds,
    params = parameter,
    early_stopping_rounds = 20,
    metric = "rmse",
    first_metric_only = TRUE
  )

  # Extract optimal number of boosting iterations
  parameter$num_iterations <- lgb_pokemon$best_iter
  rm(lgb_pokemon)

  # Train using optimal number of iterations
  lgb_pokemon <- lgb.train(
    data = dtrain,
    objective = "regression",
    valids = valids,
    params = parameter,
    eval = c("rmse", "mae")
  )

  # Make predictions for train and test dataset
  test_result$PredictedPortionRemainingHP_1 <- predict(lgb_pokemon, test_dt)
  train_result$PredictedPortionRemainingHP_1 <- predict(lgb_pokemon, train_dt)
  test_result[, PredictedPortionRemainingHP_1 := PredictedPortionRemainingHP_1 + ExpectedPortionRemainingHP_
1]
  train_result[, PredictedPortionRemainingHP_1 := PredictedPortionRemainingHP_1 + ExpectedPortionRemainingHP
_1]
  test_result[PredictedPortionRemainingHP_1 < 0, PredictedPortionRemainingHP_1 := 0]
  train_result[PredictedPortionRemainingHP_1 < 0, PredictedPortionRemainingHP_1 := 0]
  test_result[PredictedPortionRemainingHP_1 > 1, PredictedPortionRemainingHP_1 := 1]
  train_result[PredictedPortionRemainingHP_1 > 1, PredictedPortionRemainingHP_1 := 1]

  # Calculate regression metrics and compute time
  results[row, `:=`(
    num_iterations = parameter$num_iterations,
    test_rmse = MLmetrics::RMSE(test_result$PredictedPortionRemainingHP_1, test_result$PortionRemainingHP_1)
,
    test_mae = MLmetrics::MAE(test_result$PredictedPortionRemainingHP_1, test_result$PortionRemainingHP_1),
    test_r2 = MLmetrics::R2_Score(test_result$PredictedPortionRemainingHP_1, test_result$PortionRemainingHP_
1),
    train_rmse = MLmetrics::RMSE(train_result$PredictedPortionRemainingHP_1, train_result$PortionRemainingHP
_1),
    train_mae = MLmetrics::MAE(train_result$PredictedPortionRemainingHP_1, train_result$PortionRemainingHP_1
),
    train_r2 = MLmetrics::R2_Score(train_result$PredictedPortionRemainingHP_1, train_result$PortionRemaining
HP_1),
    time_stop = Sys.time()
  )]

  # Write results of training to CSV
  results[row, compute_time := difftime(time_stop, time_start, units = "secs")]
  data.table::fwrite(results[row], "data/06_models/lgb_parameter_search.csv", append = TRUE)

  rm(lgb_pokemon)
}
```