Create Features

Lucas Jamar

2020-05-31

Since our model needs to predict the outcome of the battle against opponents it has never seen before, we select 25% of Name_2 values for the validation set. Feature engineering is centred around the use of the damage calculation formula. Since we are interested in the maximum damage a player can inflict on the other, we start by taking the weather and type influences to be equal to the maximum weather and type influence available for each player. Then, by approximating the power of all moves to 80, we can calculate the damage and special damage each player can inflict on the other. The damage values are then rounded down and used to approximate how many rounds a player would need to defeat the other. The approximate number of rounds of the battle is equal to the smallest number of rounds one player would need to defeat the other. By ignoring all priority moves, we can consider that the faster player always moves first. Therefore, if the winning player is also the faster one, we assume that the defeated player will have one less chance to strike. From here, we can approximate the battle result as well as the portion of ramining HP of player 1. Since our approximations correlate well with the target variable (R2 = 0.95), we redefine the target variable to be the difference between the true remaining HP of player 1 and our approximation. This, in turn, should allow our ML model to focus on the points that are difficult to explain with our approximation. Finally, we calculate some ratios to create features for our model.

Read in combined data

```
library(data.table)

dt <- data.table::fread("data/03_primary/pokemon.csv")
```

Use 25% of all opponent pokemons for test data.

```
all_opponents <- dt[Set == "train", unique(Name_2)]
set.seed(12345)
test_opponents <- sample(all_opponents, size = floor(length(all_opponents) * 0.25))
dt[Name_2 %in% test_opponents, Set := "test"]
```

Create target variable & battle outcome class.

```
dt[, PortionRemainingHP_1 := BattleResult / HP_1]
dt[PortionRemainingHP_1 < 0, PortionRemainingHP_1 := 0]
dt[PortionRemainingHP_1 == 0, BattleOutcome := "Defeated"]
dt[PortionRemainingHP_1 == 1, BattleOutcome := "Undamaged"]
dt[is.na(BattleOutcome), BattleOutcome := "Damaged"]
```

Find the maximum strength per player.

```
dt[, MaxStrength_1 := pmax(Strength_1a_2a, Strength_1a_2b, Strength_1b_2a, Strength_1b_2b, na.rm = TRUE)]
dt[, MaxStrength_2 := pmax(Strength_2a_1a, Strength_2a_1b, Strength_2b_1a, Strength_2b_1b, na.rm = TRUE)]
```

Create single type class.

```
dt[Type_1b != "", Type_1 := paste(Type_1a, Type_1b, sep = " - ")]
dt[Type_2b != "", Type_2 := paste(Type_2a, Type_2b, sep = " - ")]
dt[Type_1b == "", Type_1 := Type_1a]
dt[Type_2b == "", Type_2 := Type_1a]
```

Normally, the weather has no influence.

```
dt[, WeatherInfluence_1 := 1]
dt[, WeatherInfluence_2 := 1]
```

If there is a positive weather influence available, use to player's advantage.

```
dt[WeatherAndTime == "Rain" & Type_1 %like% "Water", WeatherInfluence_1 := 1.5]
dt[WeatherAndTime == "Rain" & Type_2 %like% "Water", WeatherInfluence_2 := 1.5]
dt[WeatherAndTime == "Sunshine" & Type_1 %like% "Fire", WeatherInfluence_1 := 1.5]
dt[WeatherAndTime == "Sunshine" & Type_2 %like% "Fire", WeatherInfluence_2 := 1.5]
```

If the only type has a negative weather influence, it becomes a disadvantage.

```
dt[WeatherAndTime == "Sunshine" & Type_1 == "Water", WeatherInfluence_1 := 0.5]
dt[WeatherAndTime == "Sunshine" & Type_2 == "Water", WeatherInfluence_2 := 0.5]
dt[WeatherAndTime == "Rain" & Type_1 == "Fire", WeatherInfluence_1 := 0.5]
dt[WeatherAndTime == "Rain" & Type_2 == "Fire", WeatherInfluence_2 := 0.5]
```

On average, the random influence generator has a value 0.925. We also use maximum strength as type effectiveness.

```
dt[, Modifier_1 := WeatherInfluence_1 * 0.925 * MaxStrength_1]
dt[, Modifier_2 := WeatherInfluence_2 * 0.925 * MaxStrength_2]
```

Calculate normal damage and special damage. The most frequent move power is 80

```
dt[, Damage_1 := ((((2 * Level_1 / 5) + 2) * 80 * Attack_1 / Defense_2) / 50 + 2) * Modifier_1]
dt[, Damage_2 := ((((2 * Level_2 / 5) + 2) * 80 * Attack_2 / Defense_1) / 50 + 2) * Modifier_2]
dt[, Sp_Damage_1 := ((((2 * Level_1 / 5) + 2) * 80 * Sp_Atk_1 / Sp_Def_2) / 50 + 2) * Modifier_1]
dt[, Sp_Damage_2 := ((((2 * Level_2 / 5) + 2) * 80 * Sp_Atk_2 / Sp_Def_1) / 50 + 2) * Modifier_2]
```

Damage values are rounded down.

```
cols_to_round_down <- c("Damage_1", "Damage_2", "Sp_Damage_1", "Sp_Damage_2")
dt[, (cols_to_round_down) := lapply(.SD, floor), .SDcols = cols_to_round_down]
```

Determine max damage a player can inflict on other.

```
dt[, MaxDamage_1 := pmax(Damage_1, Sp_Damage_1)]
dt[, MaxDamage_2 := pmax(Damage_2, Sp_Damage_2)]
```

Approximate number of rounds for each player to defeat other.

```
dt[, ExpectedRounds_1_ToDefeat_2 := ceiling(HP_2 / MaxDamage_1)]
dt[, ExpectedRounds_2_ToDefeat_1 := ceiling(HP_1 / MaxDamage_2)]
```

Approximate number of rounds (minimum of both players).

```
dt[, ExpectedRounds := pmin(ExpectedRounds_1_ToDefeat_2, ExpectedRounds_2_ToDefeat_1)]
```

Approximate damage each player will inflict on other. All priorities aside, faster player moves first. If winning player is also the faster one, losing player will strike one time less.

```
dt[ExpectedRounds_1_ToDefeat_2 <= ExpectedRounds_2_ToDefeat_1 & Speed_1 > Speed_2,
   `:=` (ExpectedRemainingHP_1 = HP_1 - (ExpectedRounds - 1) * MaxDamage_2,
         ExpectedRemainingHP_2 = HP_2 - ExpectedRounds * MaxDamage_1)]
dt[ExpectedRounds_1_ToDefeat_2 >= ExpectedRounds_2_ToDefeat_1 & Speed_1 < Speed_2,
   `:=` (ExpectedRemainingHP_1 = HP_1 - ExpectedRounds * MaxDamage_2,
         ExpectedRemainingHP_2 = HP_2 - (ExpectedRounds - 1) * MaxDamage_1)]
dt[is.na(ExpectedRemainingHP_1),
   `:=` (ExpectedRemainingHP_1 = HP_1 - ExpectedRounds * MaxDamage_2,
         ExpectedRemainingHP_2 = HP_2 - ExpectedRounds * MaxDamage_1)]
dt[ExpectedRemainingHP_1 < 0, ExpectedRemainingHP_1 := 0]
dt[ExpectedRemainingHP_2 < 0, ExpectedRemainingHP_2 := 0]
```

Approximate battle result is equal to remaining HP of player with highest remaining HP

```
dt[, ExpectedBattleResult := pmax(ExpectedRemainingHP_1, ExpectedRemainingHP_2)]
```

If player 2 is expected to win, ensure approximate battle result is negative

```
dt[ExpectedRemainingHP_2 > ExpectedRemainingHP_1, ExpectedBattleResult := -ExpectedBattleResult]
dt[, ExpectedPortionRemainingHP_1 := ExpectedBattleResult / HP_1]
dt[ExpectedPortionRemainingHP_1 < 0, ExpectedPortionRemainingHP_1 := 0]
```

Calculate diff between true and expected outcome (target variable)

```
dt[, TrueMinusExpectedPortionRemainingHP_1 := PortionRemainingHP_1 - ExpectedPortionRemainingHP_1]
```

Calculate some ratios

```
dt[, RatioLevel := Level_1 / Level_2]
dt[, RatioPrice := Price_1 / Price_2]
dt[, RatioHP := HP_1 / HP_2]
dt[, RatioAttack := Attack_1 / Attack_2]
dt[, RatioDefense := Defense_1 / Defense_2]
dt[, RatioSp_Atk := Sp_Atk_1 / Sp_Atk_2]
dt[, RatioSp_Def := Sp_Def_1 / Sp_Def_2]
dt[, RatioSpeed := Speed_1 / Speed_2]
dt[, RatioWeatherInfluence := WeatherInfluence_1 / WeatherInfluence_2]
dt[, RatioDamage := Damage_1 / Damage_2]
dt[, RatioSp_Damage := Sp_Damage_1 / Sp_Damage_2]
dt[, RatioMaxDamage := MaxDamage_1 / MaxDamage_2]
dt[, RatioExpectedRounds_ToDefeat := ExpectedRounds_1_ToDefeat_2 / ExpectedRounds_2_ToDefeat_1]
dt[, RatioExpectedRemainingHP := ExpectedRemainingHP_1 / ExpectedRemainingHP_2]
dt[, RatioMaxStrength := MaxStrength_1 / MaxStrength_2]
```

Remove mainly empty columns

```
remove_columns = colnames(dt)
remove_columns = remove_columns[remove_columns %like% "_1a|_1b|_2a|_2b"]
dt[, (remove_columns) := NULL]
```

Write data to CSV

```
data.table::fwrite(dt, "data/04_features/pokemon.csv")
```