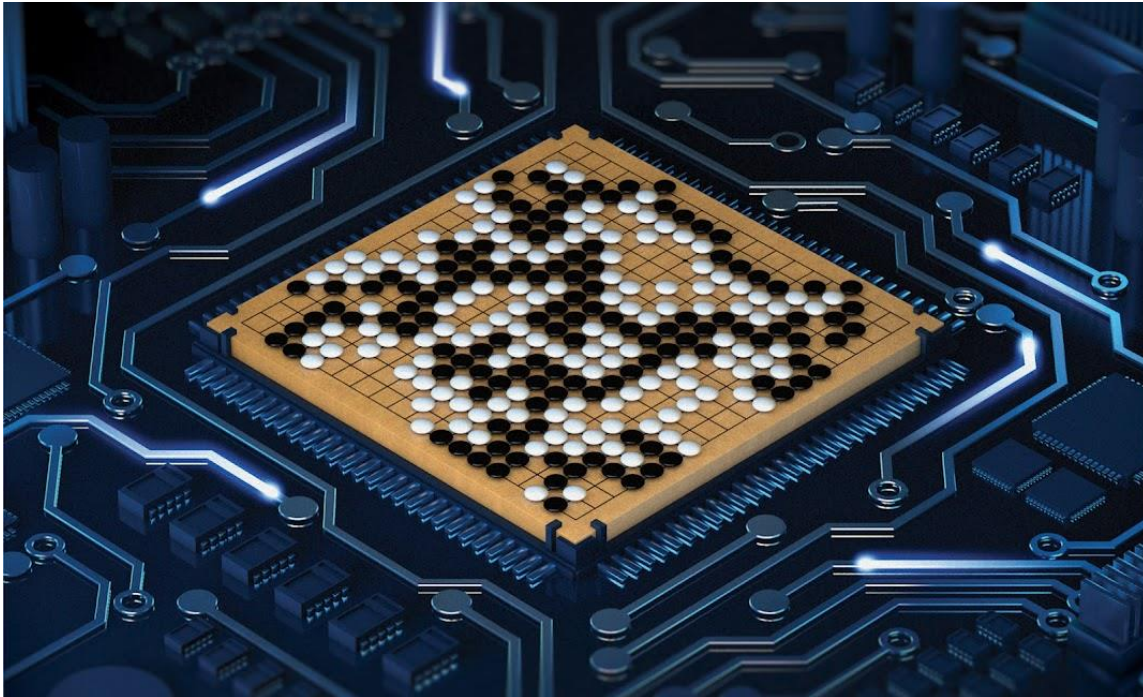


# UM – Game-Playing Strength of MCTS Variants

Predict which variants of Monte-Carlo Tree Search will perform well or poorly against each other in hundreds of board games



Team Kwak's Potatoes

강태영, 김준형, 김현동, 이나현, 이은승

# CONTENTS

---

## PART 01 Introduction

Introduction to the competition  
Target variables and what to consider

## PART 02 Data EDA & Preprocessing

Data Analysis  
Data preprocessing

## PART 03 Methodology

References & Discussion  
Design our own method

## PART 04 Our Model

현동's PCA  
태영's CatBoost  
나현's XGBoost

## PART 05 Conclusion

Public Score  
Development

# PART 01 Introduction

Introduction to the competition  
Target variables and what to consider

# 01 . Introduction to the competition

1

Kaggle competition

## UM - Game-Playing Strength of MCTS Variants

Predict which variants of Monte-Carlo Tree Search will perform well or poorly against each other in hundreds of board games



2

MCTS?

### MCTS?

A search algorithm used to develop intelligent board game players.

EX) AlphaGo



Researchers have developed various MCTS variants for games but haven't identified the best fit for specific game types.

This competition focuses on finding MCTS variants that **perform well across diverse scenarios.**

The goal is to create a model that predicts how one MCTS variant performs compared to another in different games.

## 02. The goal of the competition

1

### Goal & Evaluation

**Goal:** To predict agent 1's performance against agent 2(**Regression Problem**)

**Evaluation:** The root mean square error(**RMSE**) between the actual performance level value of agent 1 and the performance level of agent 1 we predicted.

2

### Target variable

#### The "utility\_agent1" variable

- How successful agent 1 was & How failed it was
- The difference in performance between two players



Agent 1 is dominant  
(utility\_agent1 > 0)

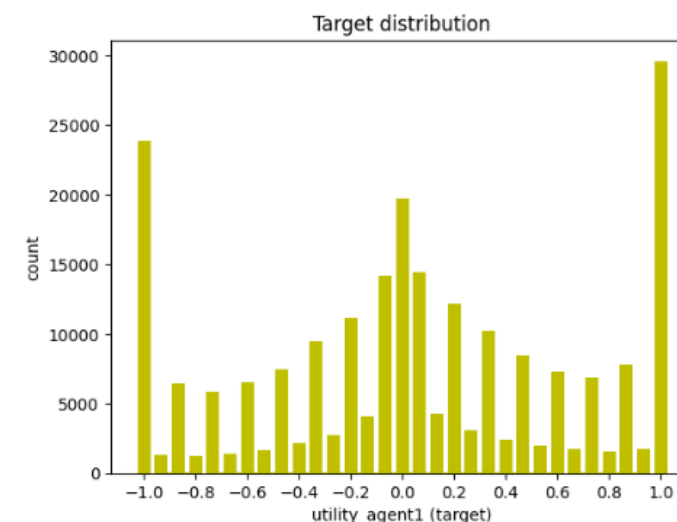


Agent 1 is equivalent  
(utility\_agent1 = 0)



Agent 1 is defeat  
(utility\_agent1 < 0)

<Target variable plot>



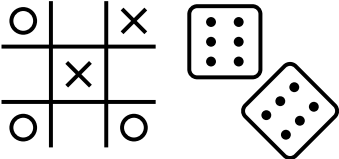
## 03 . Things to Consider in the Competition

1

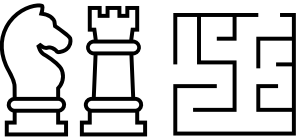
### Things to consider

#### 1. Don't just rely on certain games

<Train data>

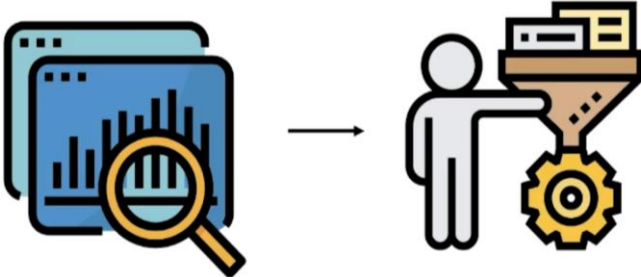


<Test data>



The test data consists of **new games** not observed in the training data.

#### 2. The amount of very large data



It's important which column you **drop** well and what **derivative** you generated.

**Choosing meaningful variables** with data EDA and preprocessing is very important!

# PART 02 Data EDA & Preprocessing

Data Analysis

Data preprocessing

# 01. Data Analysis

## 1

### Data Basic Structure

#### 2 Data

```
In [22]: train = pl.read_csv('/kaggle/input/um-game-playing-strength-of-mcts-variants/train.csv')
print('Shape before dropping columns:', train.shape)

# 상수열 제거
constant_columns = np.array(train.columns)[train.select(pl.all().n_unique() == 1).to_numpy().ravel()]
print(len(constant_columns), 'columns are constant. These will be dropped.')

# 불필요한 열 삭제
drop_columns = list(constant_columns) + ['Id']

train = train.drop(drop_columns)

train.head()
```

```
Shape before dropping columns: (233234, 814)
216 columns are constant. These will be dropped.
```

Basic training data provided in the problem : (233234, 814)  
Remove constant columns and unnecessary columns



# 01 . Data Analysis

1

## Data Basic Structure

shape: (5, 597)

Remove constant and unnecessary columns (ex. Id) : (233234, 597)

GameRulesetName	agent1	agent2	Stochastic	Asymmetric	AsymmetricForces	AsymmetricPiecesType	PlayersW
str	str	str	i64	i64	i64	i64	i64
"00Y"	"MCTS-ProgressiveHistory-0.1-MA..."	"MCTS-ProgressiveHistory-0.6-Ra..."	0	0	0	0	0
"00Y"	"MCTS-ProgressiveHistory-0.1-MA..."	"MCTS-UCB1GRAVE-0.6-NST-true"	0	0	0	0	0
"00Y"	"MCTS-ProgressiveHistory-0.1-MA..."	"MCTS-UCB1-0.1-NST-false"	0	0	0	0	0
"00Y"	"MCTS-ProgressiveHistory-0.1-MA..."	"MCTS-UCB1-0.6-NST-false"	0	0	0	0	0
"00Y"	"MCTS-ProgressiveHistory-0.1-MA..."	"MCTS-UCB1GRAVE-1.41421356237-N..."	0	0	0	0	0

### <597 column configurations>

#### 1. 5 string columns:

GameRulesetName, agent1, agent2, EnglishRules, LudRules

#### 2. 183 Float64 columns:

Continuous value columns  
ex. utility\_agent1  
(3 of these are binary)

#### 3. 409 Int64 columns:

integer value columns  
ex. num\_wins\_agent1  
(379 of these are binary)

# 01 . Data Analysis

1

## Data Basic Structure

In [15]:

```
train.select(pl.col('GameRulesetName')).to_series().value_counts(sort=True)
```

Out[15]:

shape: (1\_377, 2)

GameRulesetName	count
str	u32
"Pathway"	222
"Double_Chess"	214
"Greater_Even_Loss"	212
"Resolve"	212
"Ludus_Latrunculorum8x8_Seega_R..."	210
...	...
"58_HolesTab_Parallel_Connectio..."	72
"Bheri_Bakhri"	72
"CeelkoquyuqkoqijiFourteen_holes..."	72
"58_HolesTab_Unmarked_Suggested"	70
"Faraday"	4

- 'GameRulesName' variable

This variable represents various board game rule names.

There may be values that are not included in the test data, so do not use them as features

Use when grouping in GroupKFold =>

Minimize the impact of specific games on model performance

## 01. Data Analysis

1

## Data Basic Structure

-Player variable : The settings used in the MCTS algorithm

MCTS-<SELECTION>-<EXPLORATION\_CONST>-<PLAYOUT>-<SCORE\_BOUNDS>

<SELECTION> : Selective Policy

<EXPLORATION\_CONST> : A constant of exploration

<PLAYOUT> : Simulation method

<SCORE\_BOUNDS> : A way of calculating scores

-Number of possible combinations of each feature: 72 ( $4 \times 3 \times 3 \times 2$ )

In [17]:

```
# 교유 player 확인
train.select(pl.col('agent1', 'agent2').n_unique())
```

shape: (1, 2)

agent1	agent2
u32	u32
72	72

# plyaer 등장 횟수

```
train.select(pl.col('agent1')).to_series()
.value_counts(sort=True)
```

shape: (72, 2)

agent1	count
str	u32
"MCTS-UCB1Tuned-0.1-MAST-false"	3603
"MCTS-UCB1GRAVE-0.1-Random200-f..."	3598
"MCTS-UCB1GRAVE-0.1-MAST-false"	3579
"MCTS-UCB1GRAVE-1.41421356237-N..."	3550
"MCTS-UCB1Tuned-0.1-NST-false"	3545
...	...
"MCTS-UCB1GRAVE-0.1-NST-true"	2952
"MCTS-ProgressiveHistory-0.6-NS..."	2951
"MCTS-UCB1Tuned-0.6-NST-true"	2950
"MCTS-UCB1-1.41421356237-Random..."	2934
"MCTS-ProgressiveHistory-0.1-NS..."	2896

# 01. Data Analysis

1

## Data Basic Structure

### -Player variable

```
# Features extracted from player names
train.select(pl.col('agent1').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', 1).alias('p1_selection'),
            pl.col('agent1').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)',
2).alias('p1_exploration').cast(pl.Float32),
            pl.col('agent1').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', 3).alias('p1_playout'),
            pl.col('agent1').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', 4).alias('p1_bounds'),
            pl.col('agent2').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', 1).alias('p2_selection'),
            pl.col('agent2').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)',
2).alias('p2_exploration').cast(pl.Float32),
            pl.col('agent2').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', 3).alias('p2_playout'),
            pl.col('agent2').str.extract(r'MCTS-(.*)-(.*)-(.*)-(.*)', 4).alias('p2_bounds')
)
```

p1_selection	p1_exploration	p1_playout	p1_bounds	p2_selection	p2_exploration	p2_playout	p2_bounds
str	f32	str	str	str	f32	str	str
"ProgressiveHistory"	0.1	"MAST"	"false"	"ProgressiveHistory"	0.6	"Random200"	"false"
"ProgressiveHistory"	0.1	"MAST"	"false"	"UCB1GRAVE"	0.6	"NST"	"true"
"ProgressiveHistory"	0.1	"MAST"	"true"	"UCB1"	0.1	"NST"	"false"
"ProgressiveHistory"	0.1	"MAST"	"true"	"UCB1"	0.6	"NST"	"false"
"ProgressiveHistory"	0.1	"MAST"	"true"	"UCB1GRAVE"	1.414214	"NST"	"false"
...	...	...	...	...	...	...	...
"UCB1Tuned"	1.414214	"NST"	"false"	"ProgressiveHistory"	1.414214	"Random200"	"false"
"UCB1Tuned"	1.414214	"Random200"	"false"	"UCB1"	0.6	"MAST"	"false"
"UCB1Tuned"	1.414214	"Random200"	"false"	"UCB1GRAVE"	1.414214	"NST"	"false"
"UCB1Tuned"	1.414214	"Random200"	"false"	"UCB1GRAVE"	1.414214	"NST"	"true"
"UCB1Tuned"	1.414214	"Random200"	"true"	"UCB1Tuned"	0.6	"MAST"	"false"

### Result of Code: "New Feature Column"

Column Name	설명
p1_selection	플레이어 1(agent1)의 선택 정책.
p1_exploration	플레이어 1의 탐색 상수 (EXPLORATION_CONST). Float32로 변환됨.
p1_playout	플레이어 1의 시뮬레이션 방식.
p1_bounds	플레이어 1의 점수 계산 방식.
p2_selection	플레이어 2(agent2)의 선택 정책.
p2_exploration	플레이어 2의 탐색 상수 (EXPLORATION_CONST). Float32로 변환됨.
p2_playout	플레이어 2의 시뮬레이션 방식.
p2_bounds	플레이어 2의 점수 계산 방식.

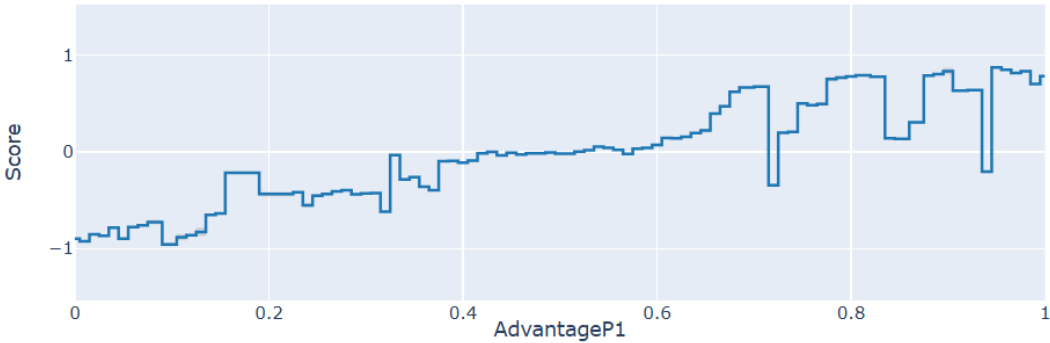
Extracted from the player names  
(agent1 and agent2) to create  
a new column

# 01. Data Analysis

2

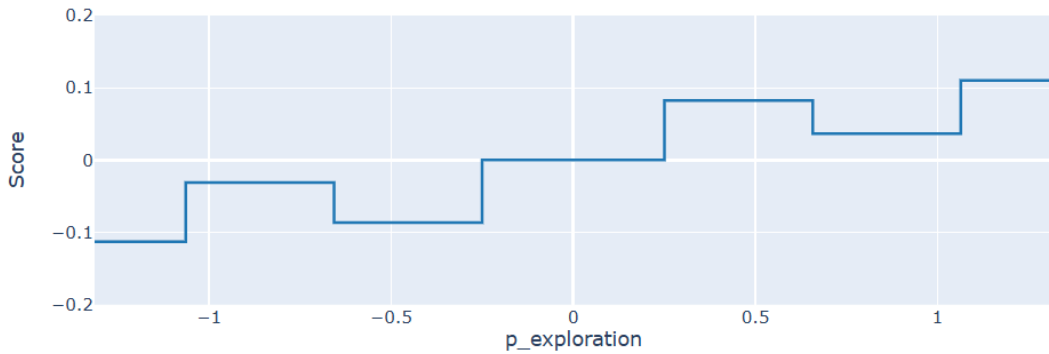
## Data EDA

The score grows with AdvantageP1



Trend of change in target value with increasing/decreasing Advantage P1 value

p\_exploration: 1.414 usually is better than 0.6, 0.6 usually is better than 0.1 (in other word

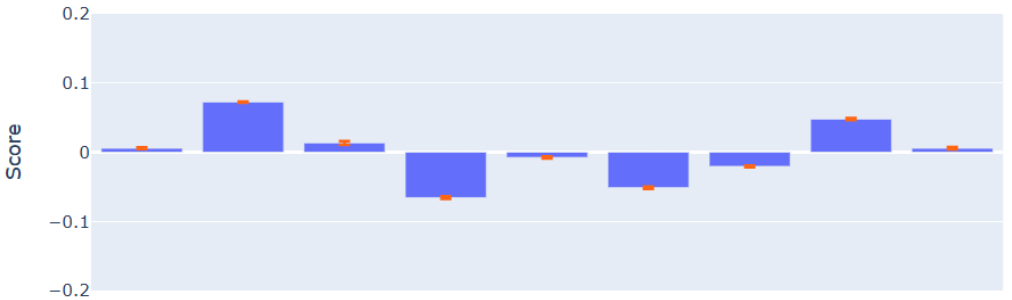


Trend of change in target value with p\_exploration  
Larger values provide better results

p\_selection: UCB1Tuned usually has an advantage against other selection strategies



p\_playout: In general, MAST is better than Random200, Random200 is better than NST



The effect of a player's MCTS strategic element (selection, playout) on the target value

## 02 . Data Preprocessing

1

### Drop Columns

```
In [23]: # 제거할 컬럼
drop_cols = [
    'Cooperation', 'Team', 'TriangleShape', 'DiamondShape', 'SpiralShape', 'StarShape',
    'SquarePyramidalShape', 'SemiRegularTiling', 'CircleTiling', 'SpiralTiling',
    'MancalaThreeRows', 'MancalaSixRows', 'MancalaCircular', 'AlquerqueBoardWithOneTriangle',
    'AlquerqueBoardWithTwoTriangles', 'AlquerqueBoardWithFourTriangles',
    'AlquerqueBoardWithEightTriangles', 'ThreeMensMorrisBoard',
    'ThreeMensMorrisBoardWithTwoTriangles', 'NineMensMorrisBoard', 'StarBoard',
    'PachisiBoard', 'Boardless', 'NumColumns', 'NumCorners', 'NumOffDiagonalDirections',
    'NumLayers', 'NumCentreSites', 'NumConvexCorners', 'NumPhasesBoard',
    'NumContainers', 'Piece', 'PieceValue', 'PieceRotation', 'PieceDirection',
    'LargePiece', 'Tile', 'NumComponentsType', 'NumDice', 'OpeningContract', 'SwapOption',
    'Repetition', 'TurnKo', 'PositionalSuperko', 'AutoMove', 'InitialRandomPlacement',
    'InitialScore', 'InitialCost', 'Moves', 'VoteDecision', 'SwapPlayersDecision',
    'SwapPlayersDecisionFrequency', 'ProposeDecision', 'ProposeDecisionFrequency',
    'PromotionDecisionFrequency', 'RotationDecision', 'RotationDecisionFrequency',
    'StepDecisionToFriend', 'StepDecisionToFriendFrequency', 'StepDecisionToEnemy',
    'SlideDecisionToEnemy', 'SlideDecisionToEnemyFrequency', 'SlideDecisionToFriend',
    'SlideDecisionToFriendFrequency', 'LeapDecision', 'LeapDecisionFrequency',
    'LeapDecisionToEmpty', 'LeapDecisionToEmptyFrequency', 'LeapDecisionToEnemy',
    'LeapDecisionToEnemyFrequency', 'HopDecisionFriendToEmpty',
    'HopDecisionFriendToEmptyFrequency', 'HopDecisionFriendToFriendFrequency',
    'HopDecisionEnemyToEnemy', 'HopDecisionEnemyToEnemyFrequency',
    'HopDecisionFriendToEnemy', 'HopDecisionFriendToEnemyFrequency',
```

1. Data with unclear or noisy meaning  
ex) shape, tiling, component, and color  
not directly related to the results of the game

2. Deduplication  
ex) drama, dramaAverage, dramaMedian  
Remove only one to prevent overfitting

3. Only apply to certain types of games or styles  
ex) ChesComponent, MancalaStyle,  
JanggiComponent

# 02 . Data Preprocessing

1

## Drop Columns

In [24]:

```
train
```

Out[24]:

	GameRulesetName	agent1	agent2	Stochastic	Asymmetric	AsymmetricForces	AsymmetricPiecesType
0	00Y	MCTS-ProgressiveHistory-0.1-MAST-false	MCTS-ProgressiveHistory-0.6-Random200-false	0	0	0	0
1	00Y	MCTS-ProgressiveHistory-0.1-MAST-false	MCTS-UCB1GRAVE-0.6-NST-true	0	0	0	0
2	00Y	MCTS-ProgressiveHistory-0.1-MAST-true	MCTS-UCB1-0.1-NST-false	0	0	0	0
3	00Y	MCTS-ProgressiveHistory-0.1-MAST-true	MCTS-UCB1-0.6-NST-false	0	0	0	0
4	00Y	MCTS-ProgressiveHistory-0.1-MAST-true	MCTS-UCB1GRAVE-1.41421356237-NST-false	0	0	0	0
...	...	...	...	...	...	...	...
233229	Zuz_Mel_7x7	MCTS-UCB1Tuned-1.41421356237-NST-false	MCTS-ProgressiveHistory-1.41421356237-Random20...	0	0	0	0
233230	Zuz_Mel_7x7	MCTS-UCB1Tuned-1.41421356237-Random200-false	MCTS-UCB1-0.6-MAST-false	0	0	0	0
233231	Zuz_Mel_7x7	MCTS-UCB1Tuned-1.41421356237-Random200-false	MCTS-UCB1GRAVE-1.41421356237-NST-false	0	0	0	0
233232	Zuz_Mel_7x7	MCTS-UCB1Tuned-1.41421356237-Random200-false	MCTS-UCB1GRAVE-1.41421356237-NST-true	0	0	0	0
233233	Zuz_Mel_7x7	MCTS-UCB1Tuned-1.41421356237-Random200-true	MCTS-UCB1Tuned-0.6-MAST-false	0	0	0	0

233234 rows × 340 columns

Final train data after  
unnneeded columns are removed  
(233324, 340)



## 02. Data Preprocessing

## 2

## Derivative Variables

```
# 새로운 컬럼 추가
train['PlayoutsPerSecond/MovesPerSecond'] = train['PlayoutsPerSecond'] / (train['MovesPerSecond'] + 1e-15)
train['EfficiencyPerPlyout'] = train['MovesPerSecond'] / (train['PlayoutsPerSecond'] + 1e-15)
train['TurnsDurationEfficiency'] = train['DurationActions'] / (train['DurationTurnsStdDev'] + 1e-15)
train['AdvantageBalanceRatio'] = train['AdvantageP1'] / (train['Balance'] + 1e-15)
train['ActionTimeEfficiency'] = train['DurationActions'] / (train['MovesPerSecond'] + 1e-15)
train['StandardizedTurnsEfficiency'] = train['DurationTurnsStdDev'] / (train['DurationActions'] + 1e-15)
train['AdvantageTimeImpact'] = train['AdvantageP1'] / (train['DurationActions'] + 1e-15)
train['DurationToComplexityRatio'] = train['DurationActions'] / (train['StateTreeComplexity'] + 1e-15)
train['NormalizedGameTreeComplexity'] = train['GameTreeComplexity'] / (train['StateTreeComplexity'] + 1e-15)
train['ComplexityBalanceInteraction'] = train['Balance'] * train['GameTreeComplexity']
train['OverallComplexity'] = train['StateTreeComplexity'] + train['GameTreeComplexity']
train['ComplexityPerPlyout'] = train['GameTreeComplexity'] / (train['PlayoutsPerSecond'] + 1e-15)
train['TurnsNotTimeouts/Moves'] = train['DurationTurnsNotTimeouts'] / (train['MovesPerSecond'] + 1e-15)
train['Timeouts/DurationActions'] = train['Timeouts'] / (train['DurationActions'] + 1e-15)
train['OutcomeUniformity/AdvantageP1'] = train['OutcomeUniformity'] / (train['AdvantageP1'] + 1e-15)
```

&lt;Create a new derivative&gt;

## 1. EfficiencyPerPlyout

$$\text{EfficiencyPerPlyout} = \frac{\text{MovesPerSecond}}{\text{PlayoutsPerSecond} + 1e - 15}$$

## 2. TurnsDurationEfficiency

$$\text{TurnsDurationEfficiency} = \frac{\text{DurationActions}}{\text{DurationTurnsStdDev} + 1e - 15}$$

## 3. AdvantageBalanceRatio

$$\text{AdvantageBalanceRatio} = \frac{\text{AdvantageP1}}{\text{Balance} + 1e - 15}$$

Create a **derivative variables** with reference to the Discussion part



# PART 03 Methodology

References & Discussion  
Design our own method

# 01. References


1

## Discussion


### Baseline Model in Discussion


By default, analyze the feature importance of the model based on preprocessed training data


#### Discussion


 Search discussions

All Owned Bookmarks

 [General] What processes are being used for Single Model improvement?  
Timothy Coffman · Last comment 3mo ago by AC

 198 Constants, 18 Nulls, ID + 4 Targets out of 812 --- 216 + 5 Features can be ignored !! [Host Conformed]  
SeshuRaju 🙏 · Last comment 3mo ago by Gaurav Rawat

 How low public lb rmse can be achieved?  
AC · Last comment 3mo ago by AC

 find a bug to improve scores  
yunsuxiaozhi · Last comment 3mo ago by bpk\_ismyname

Gradient Boosting model learning with feature selection is an effective way to achieve both performance improvement and computational efficiency

Increase the likelihood of interpreting model results and provide a clearer understanding of the core patterns of the data

It is good to use the Gradient Boosting model with **Feature Selection**

## 01. References

1

## Discussion

## Important Features

```
# Keep the good features for later
good_features = list(importance_df.query("importance > 0").index)
good_features = [f for f in good_features if f not in ['p_selection',
'p_exploration', 'p_playout', 'p_bounds']]
```

In this model, only 30% to 40% of the features are significant, and the rest are not useful.

Important features: 41% (rmse=0.446)

	importance	std	ComputationTypeId
AdvantageP1	0.161922	0.001798	Simulation
p_selection	0.037397	0.000431	Player
p_playout	0.026145	0.000240	Player
p2_selection	0.016604	0.000057	Player
p1_exploration	0.015168	0.000134	Player
p2_exploration	0.014206	0.000011	Player
p_exploration	0.011101	0.000386	Player
PlayoutsPerSecond	0.008612	0.000109	Simulation
OutcomeUniformity	0.007643	0.000432	Simulation
p1_selection	0.007164	0.000129	Player
DurationTurnsStdDev	0.006901	0.000021	Simulation
MovesPerSecond	0.004818	0.000049	Simulation
DrawFrequency	0.004357	0.000185	Simulation
Region	0.004080	0.000083	Compiler
DecisionMoves	0.004065	0.000074	Simulation
DurationMoves	0.003986	0.000319	Simulation

The most important feature is 'AdvantageP1'

The element of the player variable, which was the main variable introduced earlier, was selected as an important feature

## 02. Design our own method

1

### Modeling strategy

# Gradient Boosting

#### Concept:

Combines weak learners (e.g., decision trees) sequentially to build a strong predictive model. Each new model focuses on reducing the residuals (errors) of the previous models.

#### Strengths:

Nonlinear Relationship Learning: Captures complex, nonlinear patterns in data.

Iterative Error Reduction: Each model improves the overall accuracy by reducing prior errors.

#### How it Works:

Train the first model and compute residuals.

Train subsequent models to minimize these residuals.

Combine predictions of all models for the final output.

## 02. Design our own method

2

Model to utilize

Most Common Library:

The logo for LightGBM, featuring a teal header bar above a white box containing the text "Lightgbm".

Lightgbm

The logo for XGBoost, featuring a blue header bar above a white box containing the text "XGBoost".

XGBoost

The logo for CatBoost, featuring a dark blue header bar above a white box containing the text "CatBoost".

CatBoost

Machine Learning Techniques Learned in Class & Use the above models as well

## PART 04 Our Model

현동's PCA

태영's CatBoost

나현's XGBoost

# 01. 현동's PCA

1

## Modeling Strategy

### PCA

A technique for reducing dimensions while retaining maximum variance in data.

#### - How it Works

Standardize data → Compute covariance matrix → Find eigenvalues/vectors → Project onto principal components.

#### - Why use PCA?

- Dimensionality Reduction: Simplifies data while retaining key patterns.
- Efficiency: Speeds up computation by reducing dimensions.
- Noise Reduction: Removes irrelevant data, enhancing performance.
- Visualization: Projects data into 2D/3D for better insights.

# 01. 현동's PCA

## 2

## Model Codes



```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

numeric_features = X.select_dtypes(include=[np.number]).copy()
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_features)
```



```
from sklearn.decomposition import PCA,
IncrementalPCA

# PCA로 필요한 성분 개수 확인
pca = PCA(n_components=0.95) # 설명 분산 비율 기준
pca.fit(scaled_data)
n_components_needed = pca.n_components_ # 필요한 성분
개수

print(f"성분 개수: {n_components_needed}")

# IncrementalPCA 적용
ipca =
IncrementalPCA(n_components=n_components_needed,
batch_size=1000)
pca_data = ipca.fit_transform(scaled_data)
```

### PCA

Processes the entire dataset at once, which can be memory-intensive.  
Suitable for small to medium-sized datasets.

### Incremental PCA

Processes data in batches, making it more memory-efficient  
and ideal for large datasets.  
batch\_size=1000 was set, processing 1,000 rows of data at a time.

224 components are needed to explain 95% of the variance.



## 01. 현동's PCA

3

## Results

```
# PCA 성분별 설명 분산 비율
explained_variance_ratio =
pca.explained_variance_ratio_
print("Explained variance ratio by each component:",
explained_variance_ratio)
print("Total explained variance ratio:",
sum(explained_variance_ratio))
```

```
Explained variance ratio by each component: [0.073406  0.05550443 0.04392298 0.03820136 0.02775376 0.02495523
0.02160096 0.01811955 0.01689549 0.01560756 0.0147355  0.01399596
0.01391159 0.0126873  0.01180617 0.01144195 0.01084336 0.01036763
0.01011111 0.00988389 0.00890602 0.00876535 0.00858091 0.00846575
0.00826642 0.00806894 0.0079079  0.00774895 0.00757658 0.00732945
0.00728336 0.00705777 0.00676578 0.00662562 0.00646659 0.00639469
0.00619635 0.0061452  0.00589206 0.00572414 0.00561037 0.00546863
0.00543372 0.00533292 0.00514367 0.00500453 0.00491096 0.00488427
0.00480572 0.00476143 0.00464909 0.00459557 0.0044642  0.00439686
0.00430569 0.00420761 0.00417952 0.00416532 0.004058   0.00403531
0.00397097 0.00394932 0.00388147 0.003856   0.00379437 0.00378998
0.00370031 0.00367942 0.00359311 0.00358113 0.00352749 0.00347936
0.00344732 0.0034049  0.00337754 0.00335291 0.00330563 0.00326171
0.00321339 0.0031512  0.00313191 0.00309984 0.00308309 0.00306172
0.00304173 0.00298679 0.00294888 0.00289577 0.0028576  0.00281229
0.00279544 0.00277702 0.00274144 0.00265712 0.00264473 0.00262367
0.00259791 0.00257299 0.00254257 0.0025339  0.00246923 0.00243315
0.0024172  0.00236859 0.00236674 0.00234377 0.00231066 0.00230325
0.00224755 0.00224022 0.00221221 0.0021861  0.00214239 0.00211482
0.00210704 0.00208331 0.00207113 0.00206766 0.00203789 0.00203319
0.00199578 0.00196436 0.00196272 0.00192585 0.0019115  0.00190158
0.00189123 0.0018907  0.00186455 0.00185918 0.00182161 0.00180935
0.00179816 0.00178484 0.00176736 0.00175312 0.00173459 0.00173327
0.00171217 0.00170325 0.00169967 0.00168807 0.00168278 0.00167524
0.00164626 0.0016268  0.00160845 0.00159119 0.00158412 0.00158283
0.00154793 0.00154064 0.00150822 0.00150675 0.00149755 0.00147751
0.00144565 0.00144064 0.00143382 0.00141649 0.0013831  0.00137279
0.00136753 0.00135286 0.00132597 0.00131601 0.00130302 0.0012874
0.00127502 0.00125262 0.00124504]
Total explained variance ratio: 0.9510801716420305
```

- Explained Variance:

The principal components collectively explain 95.11% of the total variance in the dataset.

- Significance :

The most of the important information in the data is preserved while reducing its dimensionality.

## 02. 태영's CatBoost

1

### Modeling Strategy

# CatBoost

-What is CatBoost?

CatBoost is a type of Gradient Boosting algorithm designed to handle categorical data efficiently. It provides automatic encoding for categorical variables and delivers high predictive accuracy.

-Why CatBoost for this dataset?

The dataset contains many categorical features, making CatBoost ideal for leveraging its categorical data handling capabilities.

## 02. 태영's CatBoost

2

## Model code

```

# 랜덤 서치 시작
for i in range(1, n_search + 1):
    params_dict = sample_params(param_space)

    # 모델 학습
    model = CatBoostRegressor(**params_dict, cat_features=cat_features, verbose=200)
    model.fit(train_pool, eval_set=val_pool, early_stopping_rounds=50)

    # 검증 세트 RMSE 계산
    val_predictions = model.predict(X_val)
    rmse = np.sqrt(np.mean((val_predictions - y_val) ** 2))
    print(f"Search {i}: Params: {params_dict}, Validation RMSE: {rmse}")

    # 최적 모델 업데이트
    if rmse < best_rmse:
        best_rmse = rmse
        best_params = params_dict

    # 10번마다 모델 저장
    if i % 10 == 0:
        save_path = os.path.join(save_dir, f'model_search_{i}.cbm')
        model.save_model(save_path)
        print(f"Model saved at search {i}: {save_path}")

# 최적 하이퍼파라미터로 최종 모델 학습 및 저장
final_model = CatBoostRegressor(**best_params, cat_features=cat_features, verbose=200)
final_model.fit(train_pool, eval_set=val_pool)
final_model_path = os.path.join(save_dir, 'best_model.cbm')
final_model.save_model(final_model_path)
print(f"Final best model saved: {final_model_path} with RMSE: {best_rmse}")

```

## CatBoost

1. Proceed with Feature Selection with catboost base model
2. Reduce features through Feature Selection > 0
3. Proceed to RS 100 and RS 200  
→ Submission after RS200

## 02. 태영's CatBoost

3

## Results

```
# 랜덤 서치 시작
for i in range(1, n_search + 1):
    params_dict = sample_params(param_space)

    # 모델 학습
    model = CatBoostRegressor(**params_dict, cat_features=cat_features, verbose=200)
    model.fit(train_pool, eval_set=val_pool, early_stopping_rounds=50)

    # 검증 세트 RMSE 계산
    val_predictions = model.predict(X_val)
    rmse = np.sqrt(np.mean((val_predictions - y_val) ** 2))
```



## "Random Search Results"

```
{'verbose': 200, 'iterations': 1000, 'loss_function': 'RMSE',
 'l2_leaf_reg': 4, 'devices': '0', 'depth': 10, 'task_type': 'GPU',
 'random_seed': 42, 'learning_rate': 0.1}
```

```
print('Model saved at search {}: {}'.format(save_dir, save_path))
```

```
# 최적 하이퍼파라미터로 최종 모델 학습 및 저장
final_model = CatBoostRegressor(**best_params, cat_features=cat_features, verbose=200)
final_model.fit(train_pool, eval_set=val_pool)
final_model_path = os.path.join(save_dir, 'best_model.cbm')
final_model.save_model(final_model_path)
print(f"Final best model saved: {final_model_path} with RMSE: {best_rmse}")
```

## CatBoost

1. Proceed with Feature Selection with catboost base model
2. Reduce features through Feature Selection > 0
3. Proceed to RS 100 and RS 200  
→ Submission after RS200

## 02. 태영's CatBoost

## 3

## Results

```
# 랜덤 서치 시작
for i in range(1, n_search + 1):
    params_dict = sample_params(param_space)

    # 모델 학습
    model = CatBoostRegressor(**params_dict, cat_features=cat_features, verbose=200)
    model.fit(train_pool, eval_set=val_pool, early_stopping_rounds=50)

    # 검증 세트 RMSE 계산
    val_predictions = model.predict(X_val)
    rmse = np.sqrt(np.mean((val_predictions - y_val) ** 2))
    print(f"Search {i}: Params: {params_dict}, Validation RMSE: {rmse}")

    # 최적 모델 업데이트
    if rmse < best_rmse:
        best_rmse = rmse
        best_params = params_dict

    # 10번마다 모델 저장
    if i % 10 == 0:
        save_path = os.path.join(save_dir, f'best_model_{i}.cbm')
        model.save_model(save_path)
        print(f"Model saved at search {i}")

# 최적 하이퍼파라미터로 최종 모델 학습 및 저장
final_model = CatBoostRegressor(**best_params, cat_features=cat_features, verbose=200)
final_model.fit(train_pool, eval_set=val_pool)
final_model_path = os.path.join(save_dir, 'best_model.cbm')
final_model.save_model(final_model_path)
print(f"Final best model saved: {final_model_path} with RMSE: {best_rmse}")
```

```
bestTest = 0.3441999409
bestIteration = 906
```

## CatBoost

1. Proceed with Feature Selection with catboost base model
2. Reduce features through Feature Selection > 0
3. Proceed to RS 100 and RS 200  
→ Submission after RS200

## 03. 나현's XGBoost

1

### Modeling strategy



Why Use Advanced Gradient Boosting (XGBoost/LightGBM)?

- Performance: Faster training and reduced overfitting compared to standard Gradient Boosting.
- Efficiency: Optimized for high-dimensional, large datasets.
- Hyperparameter Tuning: Supports grid search for fine-tuning optimal configurations.
- Scalability: Handles complex data structures effectively.

Utilize the Advanced Gradient Boosting model, a combination of LightGBM and XGBoost

## 03. 나현's XGBoost

### 2 Reason for model selection

XGBoost

XGBoost enhances the efficiency and performance of Gradient Boosting models.

#### Advantages:

- Fast training: Utilizes parallel processing and optimized memory usage.
- Overfitting prevention: Includes regularization techniques (e.g., L1 and L2).
- Feature importance: Automatically calculates feature importance, making it easy to identify significant variables.

#### Why it fits this data:

- The dataset includes over 60 variables and likely contains non-linear relationships.
- XGBoost's precise tree-based learning and optimized hyperparameters help capture the complexity of the data.

## 03. 나현's XGBoost

### 2 Reason for model selection

LightGBM

LightGBM specializes in handling large datasets and high-dimensional features efficiently.

Advantages:

- Speed: Extremely fast training with low memory usage.
- Effective learning: Leaf-wise growth strategy enables deeper trees for capturing complex patterns.

Why it fits this data:

- The dataset's high-dimensional feature space requires efficient algorithms to process effectively.
- LightGBM handles this complexity while maintaining a faster training speed.



## 03. 나현's XGBoost

3

### Model code

```
from sklearn.ensemble import StackingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
from sklearn.ensemble import RandomForestRegressor

# XGBoost 모델 정의 xgb_model = XGBRegressor(
    n_estimators=150,
    max_depth=6,
    learning_rate=0.2,
    subsample=0.8,
    colsample_bytree=1.0,
    random_state=42
)

# LightGBM 모델 정의 lgbm_model = LGBMRegressor(
    n_estimators=150,
    max_depth=8,
    learning_rate=0.3,
    subsample=0.8,
    colsample_bytree=1.0,
    random_state=42
)
```

Hyperparameters found using Grid Search

## 03. 나현's XGBoost

### 3

### Results

Meta model: XGBoost

Hyperparameters have been simplified and used as much as possible.

```
xgb_meta = XGBRegressor(  
    n_estimators=100,  
    max_depth=3,  
    learning_rate=0.1,  
    subsample=0.8,  
    colsample_bytree=0.8,  
    random_state=42  
)  
  
# StackingRegressor 정의  
stacking_model = StackingRegressor(  
    estimators=[  
        ('xgboost', xgb_model),  
        ('lightgbm', lgbm_model),  
        # ('catboost', catboost_model)],  
    final_estimator=xgb_meta  
)  
  
# Stacking 모델 학습  
stacking_model.fit(X_train, y_train)
```

Mean Squared Error (MSE): 0.0931

Root Mean Squared Error (RMSE): 0.3051

R^2 Score: 0.7608

Model Learning Results

# PART 05 Conclusion








| Public Score  
Development

# 01. Public Score

1

Submission

We do a lot of submission and try to steadily increase the score.

Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
<div></div> <div><b><u>MCTS   final_submission(ch) - Version 1</u></b> Succeeded · Eunseung · 3d ago</div>	0.43378	0.42547	<input type="checkbox"/>
<div></div> <div><b><u>MCTS   final_submission(ch) - Version 1</u></b> Succeeded · Eunseung · 4d ago</div>	0.43383	0.42554	<input type="checkbox"/>
<div></div> <div><b><u>submission2_prediction - Version 2</u></b> Succeeded · naahyun · 4d ago</div>	0.48033	0.46718	<input type="checkbox"/>
<div></div> <div><b><u>submission2_prediction - Version 1</u></b> Succeeded · naahyun · 4d ago</div>	0.47207	0.46517	<input type="checkbox"/>
<div></div> <div><b><u>MCTS submission catboost - Version 1</u></b> Succeeded · TY · 5d ago · Notebook MCTS submission catboost   Version 1</div>	0.51822	0.50686	<input type="checkbox"/>
<div></div> <div><b><u>submission1_prediction - Version 2</u></b> Succeeded · naahyun · 5d ago</div>	0.47112	0.45773	<input checked="" type="checkbox"/>
<div></div> <div><b><u>submission1_prediction - Version 1</u></b> Succeeded · naahyun · 6d ago</div>	0.47222	0.46115	<input type="checkbox"/>

Finally, We did  
LightGBM + CatBoost

# 01. Public Score

1 Submission

285th out of 1608 teams

## UM - Game-Playing Strength of MCTS Variants

Late Submission



Overview Data Code Models Discussion Leaderboard Rules Team Submissions

280	▲ 168	seven		0.43374	162	7d
281	▲ 30	Hikari_30		0.43376		
282	▲ 453	GUO-XUN KO		0.43377		
283	▲ 46	Kea Kohv		0.43378		
284	▲ 42	hinata1119		0.43378	1	16d
285	▲ 42	Kwak's Potatoes		0.43378	14	7d

Participation

6,732 Entrants

1,865 Participants

1,608 Teams

43,846 Submissions

## 02. Development

### 1 1st place team usage model

## TabM: Advancing Tabular Deep Learning With Parameter-Efficient Ensembling

 [arXiv](#)  [Usage](#)  [Other tabular DL projects](#)

*TL;DR: TabM is a simple and powerful tabular DL architecture that efficiently imitates an ensemble of MLPs.*



Tip

For a quick overview of the paper, see [the abstract](#), [Figure 1](#) and [Page 7](#) in the [PDF](#).

## Using TabM in practice

To use TabM outside of this repository, you only need the following:

- [tabm\\_reference.py](#) : the minimal single-file implementation.
- [example.ipynb](#) : the end-to-end example of training TabM.
- [The section about hyperparameters](#).

To use `tabm_reference.py` , install the following dependencies:

```
torch>=2.0,<3  
rtdl_num_embeddings>=0.0.11,<0.1
```

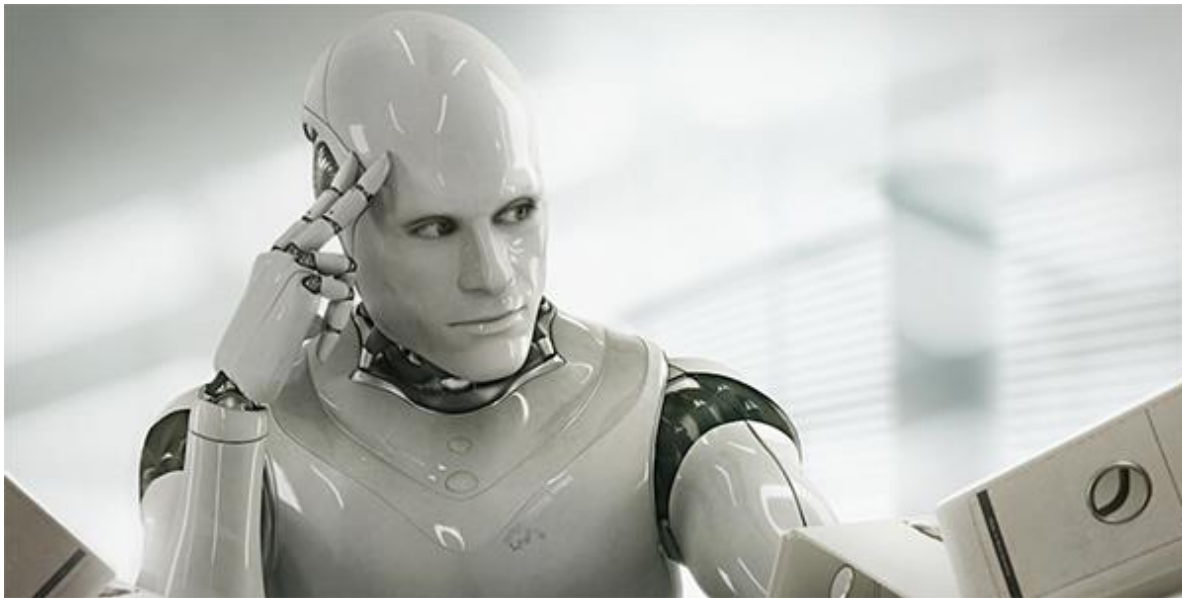


Catboost + Deeplearning

## 03. At the end of the contest

1

Impression



# Thank you for listening :)

