2023-1 데이터마이닝 기말 최종 프로젝트 보고서

컴퓨터학과 20191007 이희원

요약문

온라인 체스 게임 데이터를 활용해 게임 결과를 예측하는 연구를 수행한다. 주어진 플레이어의 움직임과 게임 상태 정보를 기반으로 등급 게임 승리 여부를 분류한다. 의사결정트리 알고리즘과 RandomForest, Support Vector Machine을 활용해 가장 정확도가 높은 모델을 찾는다. 최종 모델은 정확도와 성능 지표를 평가하여 검증되며, 게임 결과 예측의 신뢰성과 유용성을 확인한다. 이 연구는 온라인 체스 게임에서 플레이어의 승패를 예측하는 데 있어 어떤 parameter가 가장 영향을 많이 미치는지 또한 확인해본다.

Keywords: 본 과제와 관련된 핵심 키워드를 5개 이내로 작성하라.
(온라인 체스, 승패, 예측)

1. 서론

1.1 데이터마이닝을 활용한 문제 해결

본 연구의 주제는 online chess_games 데이터에서의 변수 간 "승패 여부"를 분류하는 것이다. 이번 연구에 사용하는 데이터는 온라인에서 플레이어들이 체스 게임을 진행하는 데 승리 상태, 승리한 플레이어의 말의 흑백 여부 등 게임 결과에 영향을 미치는 변수들을 이용해 의사결정 트리와 RandomForest를 이용해 게임의 결과를 예측하는 연구를 진행할 것이다.

2. 배경이론

데이터 별 parameter 분석

game_id: 진행된 게임의 고유 식별자
rated: 게임이 등급 게임인지 여부 (TRUE: 등급 게임, FALSE: 비등급 게임)
turns: 게임에서 진행된 총 턴 수
victory_status: 승리 상태 (Out of Time: 시간 초과, Resign: 항복, Mate: 체크메이트, Draw: 무승부)
winner: 승리한 플레이어 (White: 백색, Black: 흑색, Draw: 무승부)
time_increment: 플레이어의 시간 증가량
white_id: 백색 플레이어의 ID
white_rating: 백색 플레이어의 등급
black_id: 흑색 플레이어의 ID
black_rating: 흑색 플레이어의 등급
moves: 수행된 이동들
opening_code: 개방전 코드 (체스 개방전의 분류 코드)
opening_moves: 개방전에서의 수행된 이동 수
opening_fullname: 개방전의 전체 이름
opening_shortname: 개방전의 축약 이름
opening_response: 개방전에 대한 응답
opening_variation: 개방전의 변형

사용할 알고리즘 : RandomForest, Support Vector Machine

1) RandomForest

RandomForest의 개념: 여러 머신러닝 모델을 조합해 강력한 모델 설계를 위해 디자인 된 형태이다. 의사 결정 트리의 한계점인 과적합의 방지를 위해 다중 분류/예측 알고리즘을 조합해 예측 성능을 향상하는 것이다.

RandomForest 알고리즘 : 복원 추출 방식으로 데이터로부터 여러 랜덤 표본을 생성한다. 각 단계마다 랜덤으로 예측 변수들을 선택해 서브셋을 생성 후 표본에 대해 분류 트리 적합한지 확인한다. 예측 향상을 위해 각 트리로부터 얻은 예측 분류 결과를 결합한다.

분류 => 투표(voting), 예측 => 평균화를 진행한다.

2) Support Vector Machine

최대 마진 분류기(Maximal Margin Classifier)의 '그룹들은 선형 경계에 의해 구별될 수 있어야 한다.'는 한계 극복을 위해 소프트-마진을 이용한 방법이다. 분리 초평면(separating hyperplane)에서 출발해 각 모형이 갖는 한계를 극복해 발전한 분류 모형이라고 할 수 있다.

- 초평면(hyperplane) : p차원 공간에서 차원이 p - 1인 아핀(affine) 부분 공간.
- p차원 공간을 두 공간으로 나눌 수 있고, 초평면을 이용하면 초평면 식에 대입해 그 부호에 따라 특정 점이 초평면 상단, 하단 중 어디에 위치했는지 알 수 있다.
- 주어진 데이터를 완벽하게 분류하는 초평면을 정의할 수 있다면, 신규 관측치에 대해 분류 역시 가능하다.
- 초평면에 임의의 관측치 X를 대입한 결과의 부호에 따라 두 집단으로 분류할 수 있다.
- 분리 초평면의 값은 분류 결과의 정확도와 관련이 있다. 신규 관측치를 초평면에 대입했을 때 절대값이 크면 그만큼 새로운 관측치가 초평면으로부터 멀리 떨어져 있음을 의미한다.
- 하나의 분리 초평면이 존재한다면, 분리 초평면은 무수히 많이 존재할 수 있다. 그렇기에 무한개의 초평면 중 어떤 초평면을 이용할 것인지 합리적인 선택이 필요하다.
- 마진 초평면을 이용하는 방법이 있다.
- 마진 : 주어진 관측치와 초평면 사이의 수직거리를 의미, 이 거리가 최대가 되게 하는 초평면을 선택하면 합리적이고 유일한 초평면 선택할 수 있다.
- 주어진 관측치와 초평면의 수직 거리가 최대가 되는 초평면을 최대 마진 초평면이라고 한다. 이런 최대 마진 초평면을 최대 마진 초평면이라고 하고, 이런 초평면을 이용한 분류기를 최대 마진 분류기라고 한다.
- 최대 마진 분류기는 분류 경계에 있는 서포트 벡터들이 경계를 넘나들며 존재할 때 정의되지 않는다는 문제점이 있다. 또한 서포트 벡터에 대한 의존도가 절대적이라 분류 경계면에 존재하는 관측치들의 작은 변화에도 큰 영향을 받아 안정성이 떨어진다.
- 서포트 벡터 분류기 : 최대 마진 분류기의 한계를 극복하기 위해 분류 경계면에 일부 위반이 발생하더라도 다른 관측치들을 잘 분류할 수 있는 초평면을 찾는 소프트 마진(soft margin)을 적용한 방법

또한, 서포트 벡터 머신은 비선형 관계를 설명할 수 없는 벡터 분류기의 한계를 극복하기 위한 방법이다. 비선형 문제를 해결하기 위해 SVM은 커널 함수를 이용해 한 차원 높은 공간에서 초평면을 찾는다. 서포트 벡터 머신 회귀(Support Vector Machine Regression) 같은 경우, 두 그룹 사이 거리를 충분히 멀리 유지하고 마진 위반을 예방하는 것 대신 제한된 마진 오류 안에서 가능한 한 많이 마진 위에 있도록 학습하는 방법이다. GridSearchCV는 교차 검증과 하이퍼 파라미터 튜닝을 한 번에 해주는 유용한 기능이다.

- Mapping function : Input Space와 Feature Space 사이를 매핑해 주는 함수
- 커널(Kernel) : 고차원 매핑과 내적을 한 번에 해결하기 위한 개념이다. 고차원에서의 내적을 계산하는 함수이다.
- Mercer's Thorem : 커널 함수가 내적이 되기 위한 조건 => '대칭성'과 양의 준정부호성(PSD)에 대해 정의한 것을 의미한다.
- estimator : regressor, classifier 등이 사용된다.
- param_grid : 딕셔너리 형태로, 파라미터 튜닝을 위한 여러 파라미터 명과 값을 지정한다

- n_jobs : CPU 코어 사용 개수를 의미한다. (-1 지정 시 모든 코어에 사용된다.)
- refit : 최적의 하이퍼 파라미터를 찾은 뒤 그 파라미터를 입력된 estimator로 재학습한다.
- cv : cross validation, 교차 검증을 위해 분할되는 세트의 개수를 의미한다.

3. 연구 방법

3.1 데이터 전처리 및 데이터 탐색

1) 데이터 전처리

```
In [19]: from pathlib import Path
         import pandas as pd
         import numpy as np
         from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
         from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
         from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
         from sklearn.metrics import accuracy_score, confusion_matrix
         import matplotlib.pylab as plt
         import dmba
         from dmba import plotDecisionTree, classificationSummary, regressionSummary
         from sklearn import svm
         import seaborn as sns
         from matplotlib import pyplot as plt

         import warnings
         warnings.filterwarnings('ignore')
         %matplotlib inline
         !pip install --upgrade matplotlib
         !pip install seaborn
         import matplotlib.pyplot as plt
```

그림 1 필요한 함수들 import

```
In [3]: chess_df_dt = pd.read_csv('data/chess_games.csv')
        chess_df_rf = pd.read_csv('data/chess_games.csv')
        chess_df_svm = pd.read_csv('data/chess_games.csv')
        #데이터를 3개로 나눠서 진행하는 이유 : 각각 데이터의 내용이 달라질 예정이라 따로따로 진행
        #3개 다 필요한 전처리 과정은 한번에 진행
```

그림 2 가져오는 데이터 읽기

여기서 dataframe을 3개로 나눠서 가져오게 되었는데, 각각 decision tree, random forest, support vector machine 알고리즘에 사용되는 데이터이다. support vector machine 같은 경우에는 데이터 전처리 과정에서 숫자 데이터의 일부 정규화가 필요해 따로 생성하게 되었다. 원래 의사결정 트리도 같이 진행하려고 하였으나 Winner가 상태가 3개라 병합하면 성능 평가가 제대로 될 것 같지 않아 Random Forest와 Support Vector Machine만 사용한다.

```
In [5]: #결측치 분석
        chess_df_dt.info()

        print(chess_df_dt.isnull().sum())
        #opening_response와 opening_variation이 무엇인지 알아야함
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20058 entries, 0 to 20057
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   game_id            20058 non-null  int64
 1   rated              20058 non-null  bool
 2   turns              20058 non-null  int64
 3   victory_status     20058 non-null  object
 4   winner             20058 non-null  object
 5   time_increment     20058 non-null  object
 6   white_id           20058 non-null  object
 7   white_rating       20058 non-null  int64
 8   black_id           20058 non-null  object
 9   black_rating       20058 non-null  int64
 10  moves              20058 non-null  object
 11  opening_code       20058 non-null  object
 12  opening_moves      20058 non-null  int64
 13  opening_fullname   20058 non-null  object
 14  opening_shortname  20058 non-null  object
 15  opening_response   1207 non-null   object
 16  opening_variation  14398 non-null  object
dtypes: bool(1), int64(5), object(11)
```

그림 3 결측치 있는 dataFrame 확인

결측치 처리를 위해 어떤 데이터가 결측치인지 확인한다. 결측치가 있는 column은 "opening_response", "opening_variation"이었다. 각각 개방전에 대한 응답과, 개방전 변형을 의미한다.

```
In [6]: #데이터 전처리 및 정규화
        #결측치 처리하기
        # opening_response -> 결측치를 'Unknown'으로 채움
        chess_df_dt['opening_response'] = chess_df_dt['opening_response'].fillna('Unknown')
        chess_df_rf['opening_response'] = chess_df_rf['opening_response'].fillna('Unknown')
        chess_df_svm['opening_response'] = chess_df_svm['opening_response'].fillna('Unknown')
```

그림 4 opening_response 결측치 채우는 코드

결측치가 많은 opening_response같은 경우 데이터가 있는 부분을 확인해보니 개방전에 대한 응답 코드였다. 그래서 결측치 같은 경우 "알 수 없음"을 뜻하는 "Unknown"으로 결측치를 채웠다.

```
In [8]: #데이터 프레임에서 중복 제거, unique한 값 확인
        chess_df_dt['opening_variation'].unique()
```

```
Out[8]: array(['Exchange Variation', 'Kennedy Variation', 'Leonardis Variation',
                'Zukertort Variation', nan, 'Mongoose Variation',
                'Pietrowsky Defense', 'Schilling-Kostic Gambit',
                'Mieses-Kotroc Variation', 'Advance Variation', 'Knight Variation',
                '#2', 'Italian Variation', 'Two Knights Defense', '#3',
                'Anti-Fried Liver Defense', 'Bowdler Attack',
                "King's English Variation", 'Smith-Morra Gambit #2',
                'Chigorin Variation', 'Haxo Gambit', 'Marshall Defense',
                'Canal Attack', 'Central Variation', 'Tartakower Variation',
                'Dragon Variation', 'Closed Variation', 'Normal Variation',
                'Beyer Gambit', 'Ross Gambit', 'Mengarini Variation',
                'Larsen Variation', 'Lopez Countergambit', 'Kingside Fianchetto',
                "Queen's Knight", 'Mason Attack', 'Advance', 'London System',
                'La Bourdonnais Variation', 'Anglo-Lithuanian Variation',
                'Winawer Variation', 'Steinitz Countergambit', 'Giuoco Pianissimo',
                'Marshall Variation', 'Rubinstein Variation',
                'Traditional Variation', 'Old Variation', 'Semi-Tarrasch',
                "Queen's Knight Variation", 'Classical Variation',
                'Ponziani Gambit', 'Benoni-Indian Defense', 'French Variation',
                'Mikenas Variation', 'Nimzowitsch Variation', 'Maroczy Variation',
```

그림 5 opening_variation 종류 확인

opening_variation 같은 경우에는 개방전의 변형형태인 것 같아 unique()를 이용해 중복되지 않은 종류로 출력했다.

```
In [9]: #value_counts()로 개수 확인하기
        chess_df_dt['opening_variation'].value_counts()

Out[9]: #2                          797
        Exchange Variation          444
        Classical Variation         419
        Normal Variation            377
        Advance Variation           301
                                   ...
        Taimanov Variation            1
        Been-Koomen Variation         1
        Philidor Gambit               1
        Romanishin Variation #2       1
        Slav                          1
        Name: opening_variation, Length: 615, dtype: int64
```

그림 6 opening_variation 종류별 개수 분포 확인

opening_variation의 종류를 확인했을 때, 최빈값으로 결측치를 채우는 것이 좋을 것 같아 value_counts()를 이용해 각 variation 별 개수를 파악했다. #2의 개수가 가장 많아 #2로 결측값을 처리했다.

```
In [12]: #개방전의 variation의 최빈값으로 채우기
         chess_df_dt['opening_variation'] = chess_df_dt['opening_variation'].fillna('#2')
         chess_df_rf['opening_variation'] = chess_df_rf['opening_variation'].fillna('#2')
         chess_df_svm['opening_variation'] = chess_df_svm['opening_variation'].fillna('#2')
```

그림 7 opening_variation 결측치 처리

그리고 변수 중에 이름이나 속성이 비슷했던 변수가 있었다. opening_fullname과 opening_shortname 이었다. 둘 다 개방전의 이름을 의미하는데, full name인지, short name인지에 대한 구분 뿐이라 둘 중 하나를 제거해도 무방해 fullname을 제거했다.

2) 데이터 탐색

```
In [13]: #데이터 탐색
         #1. opening_name을 short나 full 중 하나만 남기기(full 삭제)
         chess_df_dt = chess_df_dt.drop(columns = ['opening_fullname'])
         chess_df_rf = chess_df_dt.drop(columns = ['opening_fullname'])
         chess_df_svm = chess_df_dt.drop(columns = ['opening_fullname'])
```

그림 8 중복 변수 제거하기

parameter별 관계를 확인하기 위해 숫자 타입의 변수로 히트맵을 그렸다.

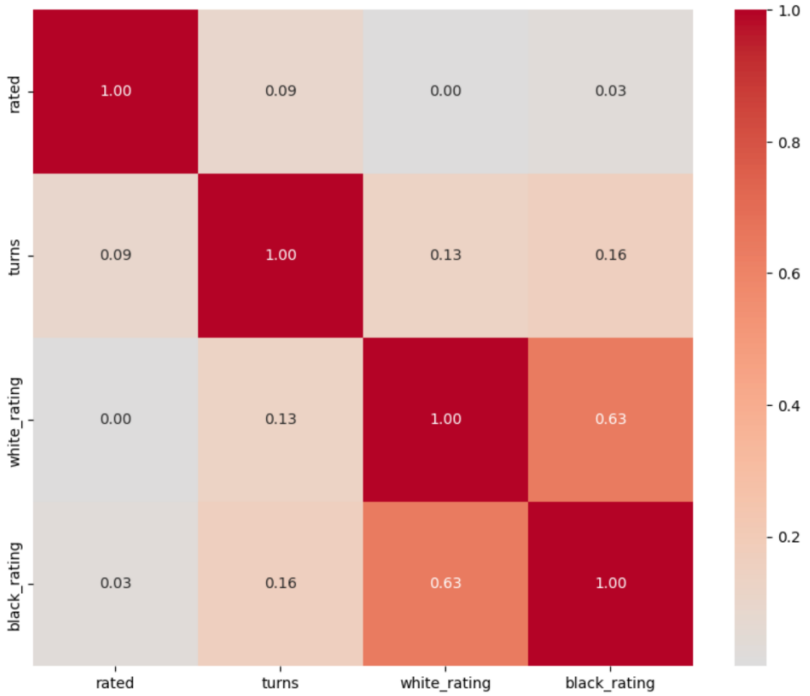

그림 9 히트맵 결과

결과를 봤을 때, black과 white의 등급 간 상관관계가 높고, 나머지는 큰 영향을 주지 않음을 알 수 있다.

의사 결정 트리나 RandomForest를 사용하는 데 정규화는 진행하지 않고, 변수들을 눈으로만 봤을 때 서로 연관이 높아보이는 변수들을 한 개만 남기고 제거를 진행한다. 단, 의사 결정 트리 같은 경우는 주로 이진 분류를 진행하는 알고리즘이라, 현재 주어진 데이터에는 예측하고자 하는 승리 상태가 White, Black, Draw의 세가지 상태라 병합을 진행한다. White,Black은 Win으로 병합하고, 그 외 Draw로 두가지 분류를 나눈다. Support Vector Machine 같은 경우 수치화된 데이터들이 필요하고, 정규화도 필요하기 때문에 데이터 전처리를 수행한다.

3.2 알고리즘 적용하기

1) Random Forest

Random Forest를 적용할 변수 10개를 찾아서 데이터 프레임에 적용한다. 적용할 column은 'moves', 'time_increment', 'rated', 'victory_status', 'turns', 'black_rating', 'white_rating', 'victory_status', 'opening_moves', 예측할 변수 'winner' 이다. (변수 설명 생략)

```
In [15]: #RandomForest

In [16]: #변수 추출하기 -> 10개로 축소
         selected_columns = ['winner', 'moves', 'time_increment', 'rated', 'victory_status', 'turns', 'black_rating',
                             'white_rating', 'victory_status', 'opening_moves']
         chess_df_rf = chess_df_rf[selected_columns]
```

그림 10 변수 축소 과정

그리고 winner를 예측하기 위해 숫자형이 아닌 변수는 인코딩이 필요하다. time_increment 같은 경우는 문자열을 숫자로 변환한다. 나머지 변수들은 pd.get_dummies를 이용해 원-핫 인코딩을 진행한다. 인코딩된 변수는 각각 범주형으로 변환된다.

```
In [17]: #encoding(숫자형 범수가 아닌 것들을 인코딩 필요함)
         #Winner
         #time_increment
         #rated
         #victory_status

In [18]: #time_increment
         # 문자열 열을 숫자로 변환 (time_increment 열)
         chess_df_rf['time_increment'] = chess_df_rf['time_increment'].str.extract('(\d+)').astype(int)
```

그림 11 time_increment 변환

```
In [19]: # 원핫인코딩을 위한 데이터 변환
         chess_df_rf = pd.get_dummies(chess_df_rf)

         # 결과 확인
         print(chess_df_rf.head())
```

```
   time_increment  rated  turns  black_rating  white_rating  opening_moves  \
0              15  False     13          1191          1500              5
1               5   True     16          1261          1322              4
2               5   True     61          1500          1496              3
3              20   True     61          1454          1439              3
4              30   True     95          1469          1523              5

   winner_Black  winner_Draw  winner_White  \
0             0            0             1
1             1            0             0
2             0            0             1
3             0            0             1
4             0            0             1

   moves_Na3 Nf6 b3 e5 e3 Nc6 c4 Be7 d4 exd4 exd4 d5 Nc2 O-O Nf3 Bg4 Be2 Ne4 Ba3 Bxa3 Nxa3 Nc3 Qd3 Nxe2 Kxe2 Re
8+ Kd2 Bxf3 gxf3 dxc4 Nxc4 Qxd4 Ne3 Rad8 Qxd4 Nxd4 Kd3 Nxf3+ Kc3 Rd2 Nd1 Ree2 a4 c5 Rc1 Ne5 Nb2 Rxb2 Rhe1 f6 f4
Nc6 Rxe2 Rxe2 Rd1 Kf7 Rd7+ Re7 Rd5 b6 a5 Nxa5 b4 cxb4+ Kxb4 Ke6 Rb5 Nc6+ Kc4 Rb7 f5+ Ke7 Rd5 a6 Kb3 b5 h4 a5 h5
a4+ Ka3 b4+ Kxa4 b3 Rc5 b2 Rxc6 b1=Q Re6+ Kf7 Rb6 Ra7+ Ra6 Rxa6#  \
0                                                  0
1                                                  0
2                                                  0
3                                                  0
4                                                  0

   ... \
0  ...
1  ...
2  ...
3  ...
4  ...
```

그림 12 나머지 변수 인코딩 결과(이하 생략)

RandomForest를 적용하기 위해서 훈련 데이터와 테스트 데이터를 나누고, 예측할 X값에는 'winner' 관련 변수를 drop하고, y값에 winner변수를 남긴다. 그리고 나서 RandomForestClassifier를 이용해 RandomForest를 진행한다.

```
X = chess_df_rf.drop(columns=['winner_Black', 'winner_Draw', 'winner_White'])
y = chess_df_rf[['winner_Black', 'winner_Draw', 'winner_White']]
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=42
RF = RandomForestClassifier(n_estimators=500, random_state=42)
RF.fit(train_X, train_y)
```

```
RandomForestClassifier(n_estimators=500, random_state=42)
```

그림 13 RandomForestClassifier 실행

변수의 중요도를 보기 위해서 feature_importances_를 이용해서 확인한다. 그리고 시각적으로 확인하기 위해 막대그래프를 그렸다. 실행결과가 제대로 나오지 않아서 코드만 첨부했다.

```
#막대 그래프
ax = df.plot(kind='barh', xerr='std', x='feature', legend=False)
ax.set_ylabel('graph')

plt.tight_layout()
plt.show()
```

그림 14 막대그래프 코드

2) Support Vector Machine

우선 Random Forest와 동일하게 변수 선택을 하고, 적용했다.

```
In [4]: selected_columns = ['winner', 'moves', 'time_increment', 'rated', 'victory_status', 'turns
                            'white_rating', 'victory_status', 'opening_moves']
        chess_df_svm = chess_df_svm[selected_columns]
```

그림 15 변수 축소해서 적용

그리고 필요한 변수들 중 turns, white_rating, black_rating에 대해 MinMaxScaler를 적용했다.

```
In [5]: # 정규화를 수행할 변수들 선택
        columns_to_normalize = ['turns', 'white_rating', 'black_rating']

        # Min-Max 정규화 객체 생성
        scaler = MinMaxScaler()

        # 정규화 수행
        normalized_data = scaler.fit_transform(chess_df_svm[columns_to_normalize])
```

그림 16 MinMaxScaler 적용

나머지 time_increment에 대한 타입 적용도 실행했다.

```
In [6]: #time_increment
        # 문자열 열을 숫자로 변환 (time_increment 열)
        chess_df_svm['time_increment'] = chess_df_svm['time_increment'].str.extract('(\d+)').asty
```

그림 17 time_increment 정규화

그리고 나서, SVM을 적용했다.

```
In [ ]: #svm 적용하기

In [7]: X = np.array(chess_df_svm[['time_increment', 'rated', 'turns'
                                   , 'black_rating', 'white_rating', 'opening_moves']])
        y = np.array(chess_df_svm['winner'])

        clf = svm.SVC(kernel='linear', C=0.01)
        clf.fit(X,y)

Out[7]: SVC(C=0.01, kernel='linear')
```

그림 18 SVM 적용 코드

4. 성능 평가

4.1 Random Forest

accuracy_score를 통해 정확도를 측정했다. 67.8%로 높은 정확도는 아닌 결과가 나왔다.

```
In [32]: # 실제값과 예측값을 비교하여 정확도 계산
         accuracy = accuracy_score(valid_y, RF.predict(valid_X))

         # 정확도 출력
         print("Accuracy:", accuracy)

         Accuracy: 0.6780907278165503
```

그림 19 RandomForest_accuracy 결과

4.2 Support Vector Machine

importance로 변수 중요score를 통해 정확도를 측정했다. 62.4%의 정확도를 보였다.(RandomForest보다 낮은 결과)

```
In [8]: #성능평가
        # 변수 중요도 확인
        importance = np.abs(clf.coef_[0])
        feature_names = ['time_increment', 'rated', 'turns', 'black_rating', 'white_rating', 'oper
        feature_importance = dict(zip(feature_names, importance))
        sorted_importance = sorted(feature_importance.items(), key=lambda x: x[1], reverse=True)
        print("Variable Importance:")
        for feature, importance in sorted_importance:
            print(f"{feature}: {importance}")

        # 정확도 측정
        accuracy = clf.score(X, y)
        print("Accuracy:", accuracy)

        Variable Importance:
        rated: 0.00032256099277816475
        opening_moves: 5.363521793810833e-06
        turns: 4.4159235130791785e-06
        time_increment: 6.924663864538161e-07
        white_rating: 4.5096385292708874e-08
        black_rating: 2.964952727779746e-08
        Accuracy: 0.6240402831787816
```

그림 20 importance와 acuuracy score(SVM)

5. 결론

RandomForest같은 경우 SVM보다 정확도는 좋게 나왔으나, 막대 그래프를 확인해보지 못해 변수의 연관성을 확인하지 못했다. SVM은 변수 중요도 결과를 보면 rated변수가 가장 높은 중요도를 가지고 있고, opening_moves, turns, time_increment, white_rating, black_rating 순으로 중요도가 낮아지는 것을 확인할 수 있다. 이번에 데이터로 사용한 온라인 체스 데이터를 봤을 때 가장 중요한 요소가 등급게임인지 아닌지로 나뉜다는 것이고, 시작할 때 움직이는 요소도 중요한 영향을 미친다는 것이다. 정확도가 좋은 모델을 뽑아내지는 못했지만, epoch나 early-stopping을 적용하면 더 좋은 알고리즘으로 발전할 수 있을 것으로 기대한다.

6. 참고문헌

https://www.kaggle.com/datasets/ulrikthygepedersen/online-chess-games?resource=download
https://www.kaggle.com/code/aymenmouffok/analyzing-chess-game-data-90-accuracy(데이터 탐색)
데이터마이닝 강의 자료(Random Forest, SVM 부분)
https://www.kaggle.com/code/mostafaa8/online-chess-games-data-analysis