텍스트이(가) 표시된 사진

자동 생성된 설명

Kwak Yu Seok 201800434

Kim Woo Yeol 201500673

Kim Jeong Yong 201400728

Choi Ji Woong 201703623

**Data Science**

**Final Report**

**1. Introduction**

1-1) **The background knowledge we need to understand the project.**  
  
-EPL: ENGLISH PREMIER LEAGUE.

The English football league is classified into 20 divisions. EPL is the highest division among them, and the second division is below it.

-UCL: UEFA Champions League.

Top class teams of every European football league participate in UCL. But each football league get different number of UCL tickets. Some leagues get 4 tickets (it means top 4 clubs get tickets), other leagues get 3 or 2 or 1 tickets. EPL will get 4 tickets to the UCL of next year.

-EPL is held on a yearly basis.

-EPL is composed of 20 teams. Every team plays 38 matches a year. There are 19 home matches and 19 away matches.

-The team which is in first place win EPL. The teams which are in first, second, third, fourth place will get a ticket to next year UCL. The teams which are in 18th, 19th, 20th place will relegate to second division next year.

-The criteria of EPL league

1) The higher points, the higher rank.

(points=3\*number of matches won + 1\* number of matches drawn+0\*number of matches lost)

2) If same points, the higher goals difference, the higher rank.

(goals difference = goals for-goals against)

3) If same points and same goals difference, the more goals for, the higher rank.

테이블이(가) 표시된 사진

자동 생성된 설명

(source: <https://www.premierleague.com/tables?co=1&se=-1&ha=-1>)

**1-2) The reason for the project selection and the Objective of the project.**

EPL (English Premier League) is one of the most famous football league. Each clubs set different objectives depending on their football capacity. Big clubs aim to win EPL, Small clubs aim not to be relegated.

3 objectives of 20 teams are

1. To win EPL (To be 1st position)

2. To get ticket to UCL (To be 1st position~4th position)

3. Not to be relegated (Not to be 18th position ~ 20th position)

But I think these 3 objectives are abstract. I think that to succeed, they need to set numerical, concrete objectives like ‘We will win 30 matches.’, ‘We will score 100 goals.’ rather than abstract objectives.

I investigated data of EPL from 1995 to 2010. (The reason why I didn’t investigate from 2011 to now is manchester city. Since 2011, Mansour bin Zayed Al Nahyan, who is of royal blood of UAE, bought Manchester City and state of EPL changed. EPL from 1995 to 2010 and EPL from 2011 to now is different. So I choose the former.)

Objective of classification, decision tree

I decided to make decision tree by classification. I made 3 decision trees.

Target of first decision tree is what is criteria of winning EPL, second one is criteria of getting ticket to UCL, third one is criteria of not being relegated.

Classifying attributes of every decision trees are same. Classifying attributes are ’total matches won’, ‘total matches drawn’ ‘total matches lost’, ‘total goals for’, ‘total goals against’.

Objective of association rule

In order to win the EPL, advance to the Champions League, or not to be demoted through the association rule, the association rule was used to find out what was most relevant.

Objective of clustering

Through clustering, we would like to analyze whether it is possible to identify the top four teams that can advance to the UCL for each year.

**1-3) information of attributes**

|  |  |  |
| --- | --- | --- |
| Attribute | type | descriptive |
| 1. classifying attribute | | |
| YEAR | Object | Year of league |
| RANK | Int64(continuous) | Rank of league |
| GF | Int64(continuous) | Total goals for |
| GA | Int64(continuous) | Total goals against |
| GD | Int64(continuous) | Total goals difference |
| WIN | Int64(continuous) | Total matches won |
| DRAW | Int64(continuous) | Total matches drawn |
| LOSE | Int64(continuous) | Total matches lost |
| POINT | Int64(continuous) | Total score |
| 2. Target attribute | | |
| EPL | Object(categorical) | Yes or no for winning the league |
| UCL | Object(categorical) | Yes or no  for entering the UCL |
| RELEGATION | Object(categorical) | Yes or no  for being relegated to second division |

II-1) To find criteria of winning EPL (by classification) +conclusion

(code)

|  |
| --- |
| in [1]  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.preprocessing import OrdinalEncoder  from sklearn.model\_selection import train\_test\_split, GridSearchCV  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  from pandas import DataFrame  import warnings  warnings.filterwarnings("ignore") |

|  |
| --- |
| in [2]  #attach csv file  EPL = pd.read\_csv('ranking.csv')  EPLEPL = pd.read\_csv('ranking.csv')  EPL  #YEAR: Year of league  #RANK: Ranking  #GF: Total goals for  #GA: Total goals against  #GD: Total goals difference  #WIN: Total number of matches won  #DRAW: Total number of matches drawn  #LOSE: Total number of matches lost  #POINT:  #EPL: Yes or no to win EPL  #UCL: Yes or no to get a ticket to UCL  #RELEGATION: Yes or no to be relegated |

|  |
| --- |
| out [2]    320 rows × 12 columns |

|  |
| --- |
| in[3]  # data preprocessing  # Because I set GF,GA, WIN, DRAW, LOSE as classifying attribute, and EPL as target attribute, I will delete attributes 'YEAR','RANK','GD','POINT','UCL', 'RELEGATION' which are not related to this decision tree.  EPL1=EPL.drop(['YEAR','RANK','GD','POINT','UCL', 'RELEGATION'],axis=1,inplace=True) |

|  |
| --- |
| in[4]  # After data preprocessing  EPL1 |

|  |
| --- |
| out [4]  GF GA WIN DRAW LOSE EPL  0 73 35 25 7 6 yes  1 66 37 24 6 8 no  2 70 34 20 11 7 no  3 52 35 18 9 11 no  4 49 32 17 12 9 no  ... ... ... ... ... ... ...  315 40 61 9 15 14 no  316 46 66 11 7 20 no  317 37 58 8 15 15 no  318 55 78 10 9 19 no  319 43 70 7 12 19 no  320 rows × 6 columns |

|  |
| --- |
| in [5]  #for classification, I set training data : test data = 7:3  feature\_cols = ['GF', 'GA', 'WIN', 'DRAW', 'LOSE']  X = EPL[feature\_cols]  y = EPL.EPL  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=100) |

|  |
| --- |
| in [6]  print("Shape of X\_train: {}".format(X\_train.shape))  print("Shape of y\_train: {}".format(y\_train.shape))  print("Shape of X\_test: {}".format(X\_test.shape))  print("Shape of y\_test: {}".format(y\_test.shape)) |

|  |
| --- |
| out [6]  Shape of X\_train: (224, 5)  Shape of y\_train: (224,)  Shape of X\_test: (96, 5)  Shape of y\_test: (96,) |

|  |
| --- |
| in [7]  #Now, I will make decision tree by using training data and test data.  # To use decision tree, I have to decide criterion ( ex)gini, entropy ) and max depth of decision tree. |

|  |
| --- |
| in [8]  param\_grid = {  'criterion':['gini', 'entropy'],  'max\_depth': list(range(2, 16)),  'min\_samples\_leaf': list(range(1, 6)),  'min\_samples\_split': list(range(2, 6)),  'random\_state':[10]  }  clf = GridSearchCV(DecisionTreeClassifier(), param\_grid, n\_jobs=-1, cv=5) |

|  |
| --- |
| in [9]  # I have to train model.  %time clf.fit(X\_train, y\_train) |

|  |
| --- |
| in [10]  clf.best\_params\_ |
|  |

|  |
| --- |
| out [10]  {'criterion': 'entropy',  'max\_depth': 6,  'min\_samples\_leaf': 1,  'min\_samples\_split': 2,  'random\_state': 10}  #Result shows that proper criterion to divide is entropy and proper max\_depth is 6. |

|  |
| --- |
| in [11]  # To make decision tree by classification, I have to check accuracy.  my\_model = clf.best\_estimator\_  my\_model.fit(X\_train, y\_train)  my\_model\_score\_train = my\_model.score(X\_train, y\_train)  my\_model\_score\_test = my\_model.score(X\_test, y\_test)  print(clf.best\_estimator\_)  print('Training data의 accuracy = ',my\_model\_score\_train)  print('Testing data의 accuracy = ',my\_model\_score\_test) |

|  |
| --- |
| out [11]  DecisionTreeClassifier(criterion='entropy', max\_depth=6, random\_state=10)  Training data의 accuracy = 1.0  Testing data의 accuracy = 0.96875  # Enough accuracy |

|  |
| --- |
| in [12]  # To make decision tree, I have to find importance of every attributes. Dividing proceed by importance of attributes.  features = pd.DataFrame(X.columns.tolist())  features.columns = ['Features']  fi = pd.DataFrame(my\_model.feature\_importances\_)  fi.columns = [‘Importance']  fea\_imp = pd.concat([features, fi], axis=1)  fea\_imp.sort\_values(by='Value', ascending = False) |

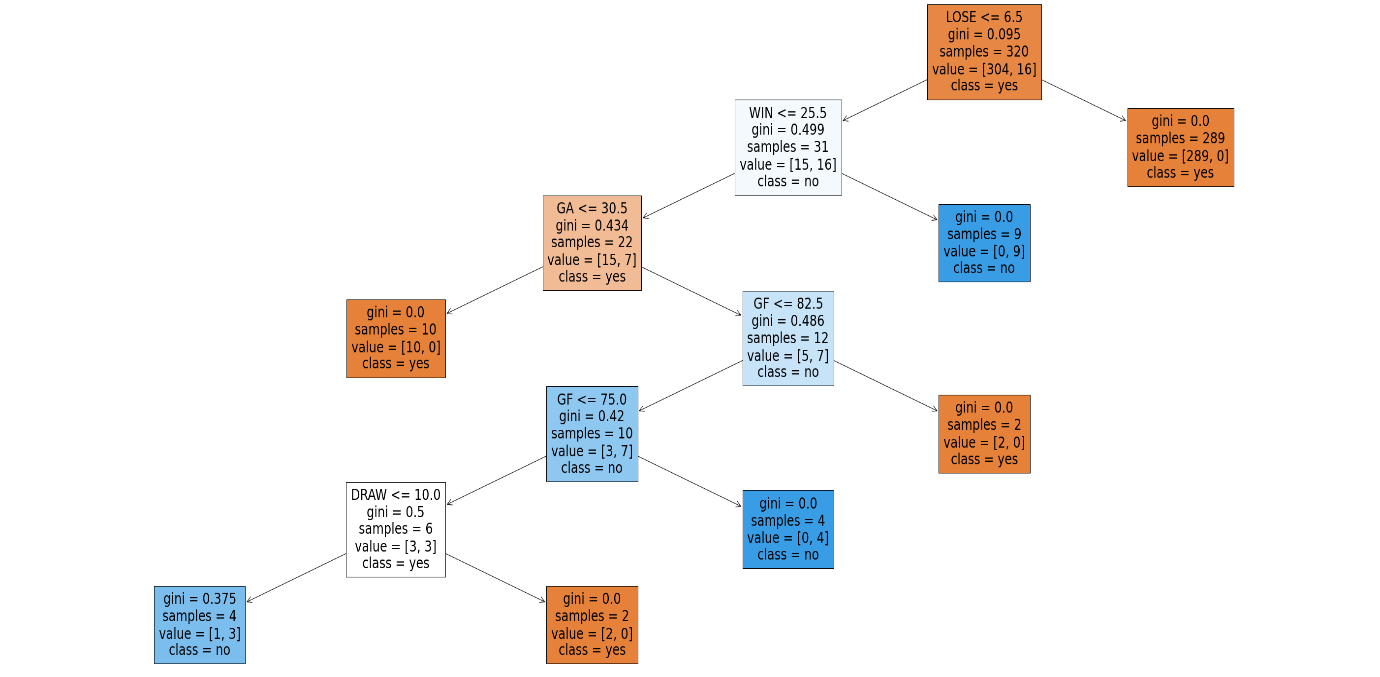
|  |
| --- |
| out [12]  Features Importance  4 LOSE 0.730405  2 WIN 0.147438  1 GA 0.076694  0 GF 0.045463  3 DRAW 0.000000  # I will divide tree by this order. Max depth is 6. So maybe every attributes are used. |

|  |
| --- |
| in [13]  # It’s time to make decision tree.  from sklearn.tree import DecisionTreeClassifier  treeclf = DecisionTreeClassifier(max\_depth=6, random\_state=1)  treeclf.fit(X, y)  from sklearn import tree  text\_representation = tree.export\_text(treeclf)  print(text\_representation)  #feature\_0: Total goals for(GF)  #feature\_1: Total goals against(GA)  #feature\_2: Total number of matches won (WIN)  #feature\_3: Total number of matches drawn (DRAW)  #feature\_4: Total number of matches lost (LOSE) |

|  |
| --- |
| out [13]  |--- feature\_4 <= 6.50  | |--- feature\_2 <= 25.50  | | |--- feature\_1 <= 30.50  | | | |--- class: no  | | |--- feature\_1 > 30.50  | | | |--- feature\_0 <= 82.50  | | | | |--- feature\_0 <= 75.00  | | | | | |--- feature\_3 <= 10.00  | | | | | | |--- class: yes  | | | | | |--- feature\_3 > 10.00  | | | | | | |--- class: no  | | | | |--- feature\_0 > 75.00  | | | | | |--- class: yes  | | | |--- feature\_0 > 82.50  | | | | |--- class: no  | |--- feature\_2 > 25.50  | | |--- class: yes  |--- feature\_4 > 6.50  | |--- class: no  # It’s just a frame of decision tree, not complete one. |

|  |
| --- |
| in [14]  #Now it’s time to make complete decision tree.  with open("decision\_tree.log", "w") as fout:  fout.write(text\_representation)  fig = plt.figure(figsize=(50,20))  \_ = tree.plot\_tree(treeclf,  feature\_names=feature\_cols,  class\_names=EPL.EPL.astype(str),  filled=True) |

out [14]



Conclusion (to win EPL)

Among 38 matches,

1. If lose <=6.5, win <= 25.5, GA<=30.5, fail to win EPL.

2. If lose <=6.5, win <= 25.5, GA>30.5, GF<=75, draw<=10, win EPL.

3. If lose <=6.5, win <= 25.5, GA>30.5, GF<=75, draw>=10, fail to win EPL.

4. If lose <=6.5, win <= 25.5, GA>30.5, 75<GF<=82.5, win EPL.

5. If lose <=6.5, win <= 25.5, GA>30.5, GF>82.5, fail to win EPL.

(In 4 and 5, 75<GF<=82.5, win EPL. But GF>82.5, fail to win EPL 🡪 It is strange. It is because GF to win EPL is relative criteria, not absolute criteria.)

6. If lose <=6.5, win > 25.5, win EPL.

7. If lose > 6.5, fail to win EPL

**Do Association Rule for EPL + Conclusion**

[1]

import pandas as pd

import numpy as np

from pandas import DataFrame

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

[2]

EPL = pd.read\_csv('Downloads/ranking.csv')

EPL.head()

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

[3]

EPL1=EPL.drop(['YEAR','RANK','GD','POINT','RELEGATION','UCL'],axis=1,inplace=True)

EPL.head()

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

# drop the columns that are not relative

[4]

epl\_copy = EPL.copy() #copy the dataset

epl\_copy.head()

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

[5]

epl\_copy['GF']=pd.qcut(epl\_copy['GF'],q=4,labels=['gfq1','gfq2','gfq3','gfq4'])

epl\_copy['GA']=pd.qcut(epl\_copy['GA'],q=4,labels=['gaq1','gaq2','gaq3','gaq4'])

epl\_copy['WIN']=pd.qcut(epl\_copy['WIN'],q=4,labels=['winq1','winq2','winq3','winq4'])

epl\_copy['DRAW']=pd.qcut(epl\_copy['DRAW'],q=4,labels=['drawq1','drawq2','drawq3','drawq4'])

epl\_copy['LOSE']=pd.qcut(epl\_copy['LOSE'],q=4,labels=['loseq1','loseq2','loseq3','loseq4'])

#divide the quartile for each columns

[6]

eplvalue = epl\_copy.values.tolist()

eplvalue

테이블이(가) 표시된 사진

자동 생성된 설명

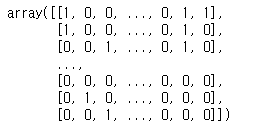
‘Yes’ > Win the EPL

[7]

te = TransactionEncoder()

te\_ary = te.fit(eplvalue).transform(eplvalue)

te\_ary.astype('int')



#encode for assoiciation rule

[8]

df = pd.DataFrame(te\_ary, columns=te.columns\_)

frequent\_itemsets = apriori(df, min\_support=0.05, use\_colnames=True)

frequent\_itemsets

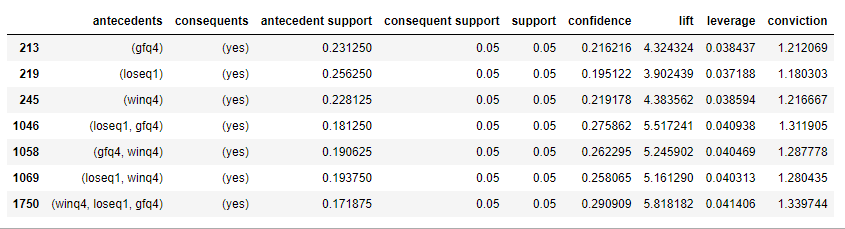
테이블이(가) 표시된 사진

자동 생성된 설명

[9]

rules = association\_rules(frequent\_itemsets, metric='support', min\_threshold = 0.05)

rules[rules['consequents'] == {'yes'}]



**Conclusion**

In this result, we found that first factor that win the EPL is Win, Lose, Goal for combination and Second is Lo

II-1) To find criteria of getting ticket to UCL (by classification)

+ Conclusion

|  |
| --- |
| in [1]  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.preprocessing import OrdinalEncoder  from sklearn.model\_selection import train\_test\_split, GridSearchCV  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  from pandas import DataFrame  import warnings  warnings.filterwarnings("ignore") |

|  |
| --- |
| in [2]  #attach csv file  EPL = pd.read\_csv('ranking.csv')  EPL  #YEAR: Year of league  #RANK: Ranking  #GF: Total goals for  #GA: Total goals against  #GD: Total goals difference  #WIN: Total number of matches won  #DRAW: Total number of matches drawn  #LOSE: Total number of matches lost  #POINT:  #EPL: Yes or no to win EPL  #UCL: Yes or no to get a ticket to UCL  #RELEGATION: Yes or no to be relegated |

|  |
| --- |
| out [2]    320 rows × 12 columns |

|  |
| --- |
| in[3]  #data preprocessing  # Because I set GF,GA, WIN, DRAW, LOSE as classifying attribute, and UCL as target attribute, I will delete attributes 'YEAR','RANK','GD','POINT','EPL', 'RELEGATION' which are not related to this decision tree.  EPL1=EPL.drop(['YEAR', 'RANK', 'GD', 'POINT', 'EPL', 'RELEGATION'],axis=1,inplace=True) |

|  |
| --- |
| in[4]  #After data preprocessing  EPL1 |

|  |
| --- |
| out [4]  GF GA WIN DRAW LOSE UCL  0 73 35 25 7 6 yes  1 66 37 24 6 8 yes  2 70 34 20 11 7 yes  3 52 35 18 9 11 yes  4 49 32 17 12 9 no  ... ... ... ... ... ... ...  315 40 61 9 15 14 no  316 46 66 11 7 20 no  317 37 58 8 15 15 no  318 55 78 10 9 19 no  319 43 70 7 12 19 no  320 rows × 6 columns |

|  |
| --- |
| in [5]  #for classification, I set training data : test data = 7:3  feature\_cols = ['GF', 'GA', 'WIN', 'DRAW', 'LOSE']  X = EPL[feature\_cols]  y = EPL.UCL  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=100) |

|  |
| --- |
| in [6]  print("Shape of X\_train: {}".format(X\_train.shape))  print("Shape of y\_train: {}".format(y\_train.shape))  print("Shape of X\_test: {}".format(X\_test.shape))  print("Shape of y\_test: {}".format(y\_test.shape)) |

|  |
| --- |
| out [6]  Shape of X\_train: (224, 5)  Shape of y\_train: (224,)  Shape of X\_test: (96, 5)  Shape of y\_test: (96,) |

|  |
| --- |
| in [7]  #Now, I will make decision tree by using training data and test data.  # To use decision tree, I have to decide criterion ( ex)gini, entropy ) and max depth of decision tree. |

|  |
| --- |
| in [8]  param\_grid = {  'criterion':['gini', 'entropy'],  'max\_depth': list(range(2, 16)),  'min\_samples\_leaf': list(range(1, 6)),  'min\_samples\_split': list(range(2, 6)),  'random\_state':[10]  }  clf = GridSearchCV(DecisionTreeClassifier(), param\_grid, n\_jobs=-1, cv=5) |

|  |
| --- |
| in [9]  # I have to train model.  %time clf.fit(X\_train, y\_train) |

|  |
| --- |
| in [10]  clf.best\_params\_ |

|  |
| --- |
| out [10]  {'criterion': 'gini',  'max\_depth': 3,  'min\_samples\_leaf': 1,  'min\_samples\_split': 2,  'random\_state': 10}  #Result shows that proper criterion to divide is gini and proper max\_depth is 3. |

|  |
| --- |
| in [11]  # To make decision tree by classification, I have to check accuracy.  my\_model = clf.best\_estimator\_  my\_model.fit(X\_train, y\_train)  my\_model\_score\_train = my\_model.score(X\_train, y\_train)  my\_model\_score\_test = my\_model.score(X\_test, y\_test)  print(clf.best\_estimator\_)  print('Training data의 accuracy = ',my\_model\_score\_train)  print('Testing data의 accuracy = ',my\_model\_score\_test) |

|  |
| --- |
| out [11]  DecisionTreeClassifier(max\_depth=3, random\_state=10)  Training data의 accuracy = 0.9910714285714286  Testing data의 accuracy = 0.96875  # Enough accuracy |

|  |
| --- |
| in [12]  # To make decision tree, I have to find importance of every attributes. Dividing proceed by importance of attributes.  features = pd.DataFrame(X.columns.tolist())  features.columns = ['Features']  fi = pd.DataFrame(my\_model.feature\_importances\_)  fi.columns = ['Importance']  fea\_imp = pd.concat([features, fi], axis=1)  fea\_imp.sort\_values(by='Value', ascending = False) |

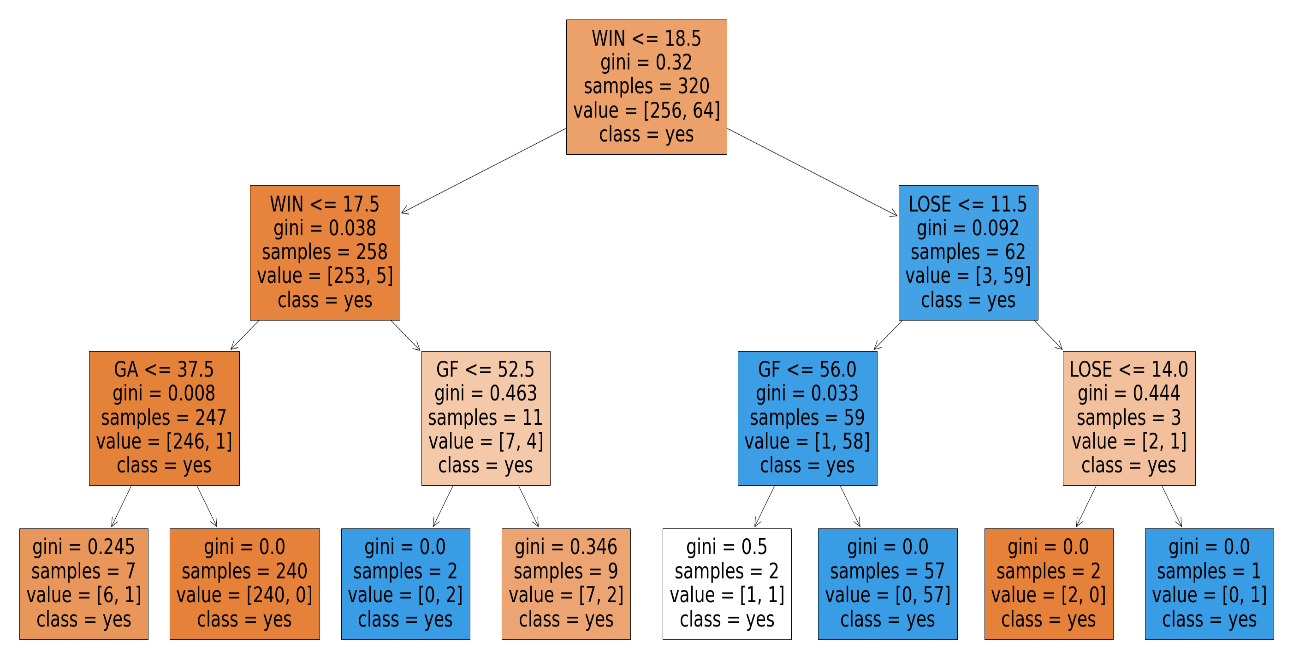
|  |
| --- |
| out [12]  Features Importance  2 WIN 0.909957  0 GF 0.086043  1 GA 0.004000  3 DRAW 0.000000  4 LOSE 0.000000  # I will divide tree by this order. Max depth is 3.  So maybe attributes ‘WIN’, ‘GF’, and ‘GA’ are used. |

|  |
| --- |
| in [13]  # It’s time to make decision tree.  from sklearn.tree import DecisionTreeClassifier  treeclf = DecisionTreeClassifier(max\_depth=3, random\_state=1)  treeclf.fit(X, y)  from sklearn import tree  text\_representation = tree.export\_text(treeclf)  print(text\_representation)  #feature\_0: Total goals for(GF)  #feature\_1: Total goals against(GA)  #feature\_2: Total number of matches won (WIN)  #feature\_3: Total number of matches drawn (DRAW)  #feature\_4: Total number of matches lost (LOSE) |

|  |
| --- |
| out [13]  |--- feature\_2 <= 18.50  | |--- feature\_2 <= 17.50  | | |--- feature\_1 <= 37.50  | | | |--- class: no  | | |--- feature\_1 > 37.50  | | | |--- class: no  | |--- feature\_2 > 17.50  | | |--- feature\_0 <= 52.50  | | | |--- class: yes  | | |--- feature\_0 > 52.50  | | | |--- class: no  |--- feature\_2 > 18.50  | |--- feature\_4 <= 11.50  | | |--- feature\_0 <= 56.00  | | | |--- class: no  | | |--- feature\_0 > 56.00  | | | |--- class: yes  | |--- feature\_4 > 11.50  | | |--- feature\_4 <= 14.00  | | | |--- class: no  | | |--- feature\_4 > 14.00  | | | |--- class: yes  # It’s just a frame of decision tree, not complete one. |

|  |
| --- |
| in [14]  #Now it’s time to make complete decision tree.  with open("decision\_tree.log", "w") as fout:  fout.write(text\_representation)  fig = plt.figure(figsize=(50,20))  \_ = tree.plot\_tree(treeclf,  feature\_names=feature\_cols,  class\_names=EPL.UCL.astype(str),  filled=True) |

output [14]



Conclusion (to get a ticket to UCL)

Among 38 matches,

1. WIN<=17.5 - fail to UCL

2. 17.5<Win<=18.5, GF<=52.5 - to UCL

3. 17.5<Win<=18.5, GF>=52.5 - fail to UCL

(In 2 and 3, GF<=52.5 to UCL, but GF>=52.5 fail to UCL 🡪 It is strange. It is because GF to UCL is relative criteria, not absolute criteria.)

4. WIN > 18.5, LOSE<=11.5, GF<=56 – fail to UCL

5. WIN > 18.5, LOSE<=11.5, GF>56 – to UCL

6. WIN > 18.5, 11.5< LOSE<=14 – fail to UCL

7. WIN > 18.5, LOSE>14 – to UCL

(In 6 and 7, 11.5< LOSE<=14 fail to UCL, but LOSE>14 to UCL 🡪 It is strange. It is because LOSE to UCL is relative criteria, not absolute criteria.)

**Do Association Rule for UCL**

[1]

import pandas as pd

import numpy as np

from pandas import DataFrame

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

[2]

EPL = pd.read\_csv('Downloads/ranking.csv')

EPL.head()

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

[3]  
EPL1=EPL.drop(['YEAR','RANK','GD','POINT','EPL','RELEGATION'],axis=1,inplace=True)

EPL.head()

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

# drop the columns that are not relative

[4]

epl\_copy = EPL.copy()

epl\_copy.head()

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

#copy the dataset

[5]

epl\_copy['GF']=pd.qcut(epl\_copy['GF'],q=4,labels=['gfq1','gfq2','gfq3','gfq4'])

epl\_copy['GA']=pd.qcut(epl\_copy['GA'],q=4,labels=['gaq1','gaq2','gaq3','gaq4'])

epl\_copy['WIN']=pd.qcut(epl\_copy['WIN'],q=4,labels=['winq1','winq2','winq3','winq4'])

epl\_copy['DRAW']=pd.qcut(epl\_copy['DRAW'],q=4,labels=['drawq1','drawq2','drawq3','drawq4'])

epl\_copy['LOSE']=pd.qcut(epl\_copy['LOSE'],q=4,labels=['loseq1','loseq2','loseq3','loseq4'])

#divide the quartile for each columns

[6]

eplvalue = epl\_copy.values.tolist()

eplvalue

테이블이(가) 표시된 사진

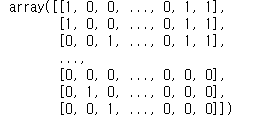
자동 생성된 설명

[7]

te = TransactionEncoder()

te\_ary = te.fit(eplvalue).transform(eplvalue)

te\_ary.astype('int')



#encode for assoiciation rule

[8]

df = pd.DataFrame(te\_ary, columns=te.columns\_)

frequent\_itemsets = apriori(df, min\_support=0.1, use\_colnames=True)

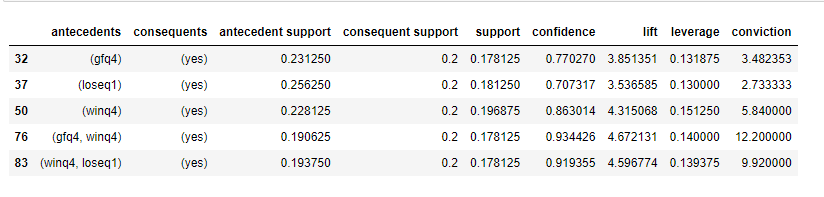
frequent\_itemsets

테이블이(가) 표시된 사진

자동 생성된 설명

[9]  
rules = association\_rules(frequent\_itemsets, metric='support', min\_threshold = 0.17)

rules[rules['consequents'] == {'yes'}]



# ‘Yes’ is success to advanced to UCL

**Conclusion**

In this result, we found that Win, Goal for combination is first factor at UCL and win, combination is second factor of UCL.

**Clustering for UCL + Conclusion**

we analyzed whether it is possible to identify teams that could distinguish to UCL through clustering.

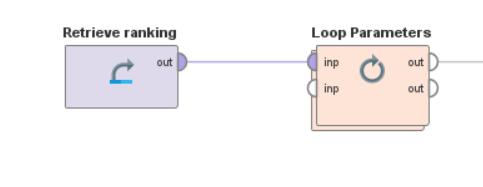
For k-means clustering, an appropriate k value was derived using the elbow method using a loop parameter.

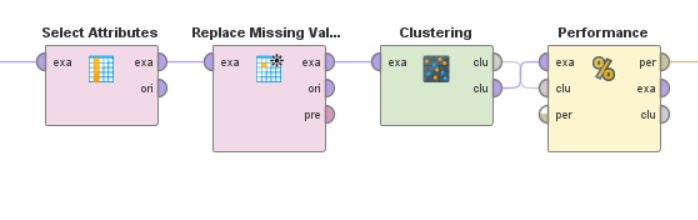
The attribute used were GA, GD, GF, WIN, DRAW, LOSE, POINT, and the polynomial data, EPL, RELEGATION, UCL, and YEAR, were excluded.

[input]

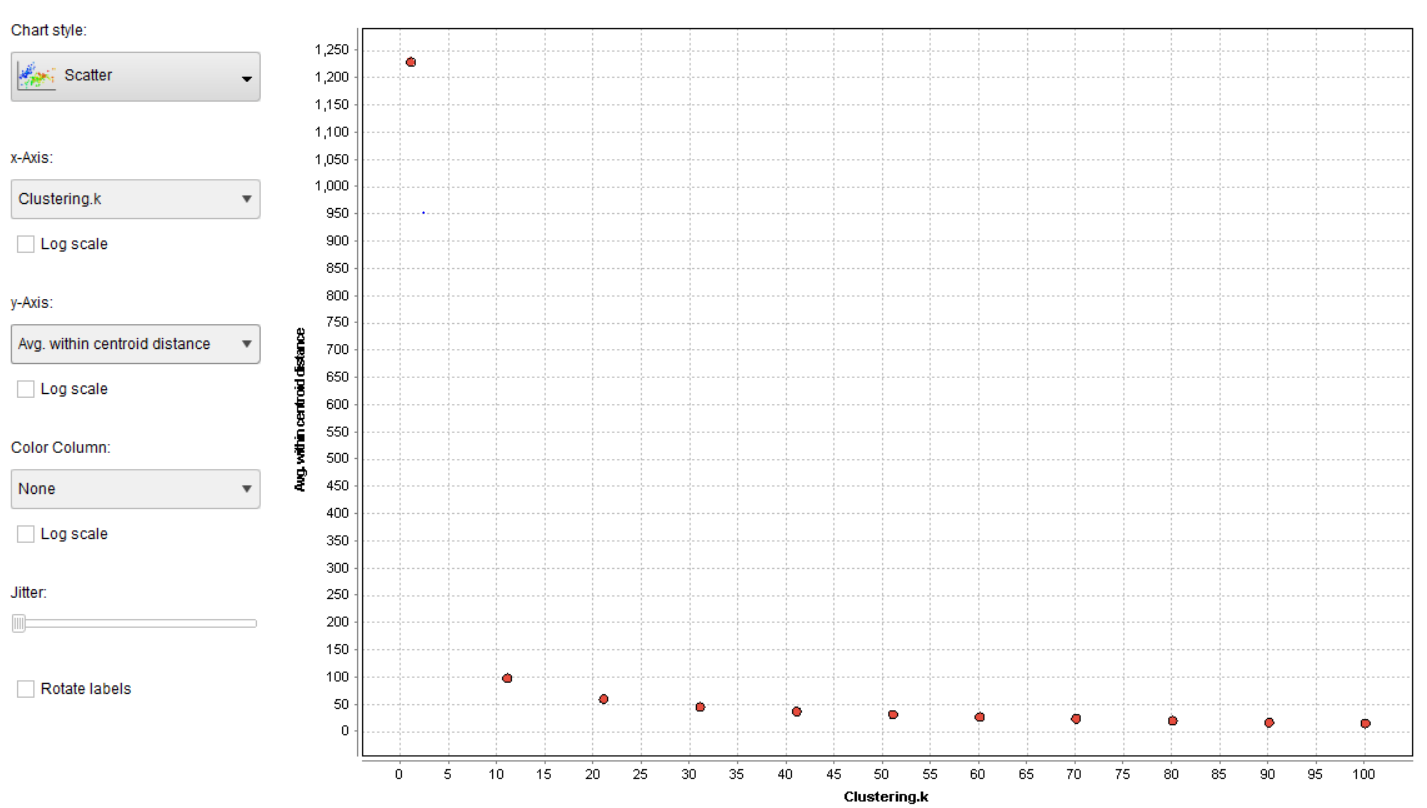
텍스트이(가) 표시된 사진

자동 생성된 설명





[output]

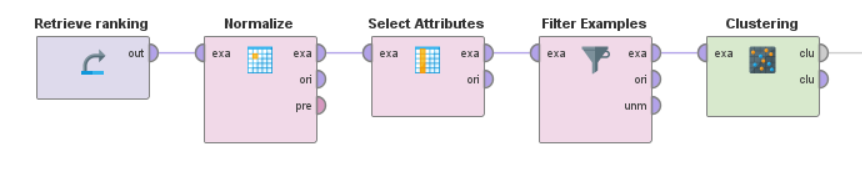


As a result, we estimated an appropriate k value of 11.

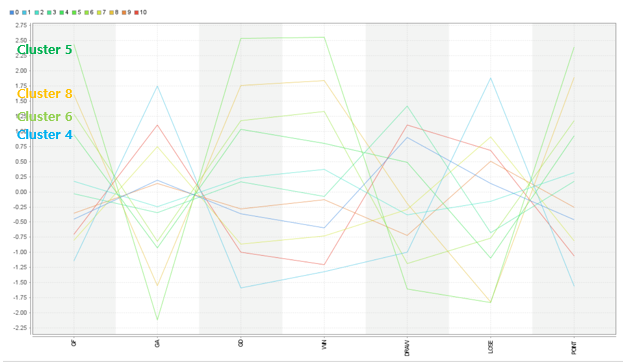
After processing the data through normalization, we proceeded with clustering using the same attributes

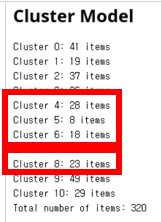
[input]

텍스트이(가) 표시된 사진

자동 생성된 설명

[output]



테이블이(가) 표시된 사진

자동 생성된 설명

This is our k=11 clustering result, of which we extracted the 77 data from the top points and obtained a figure close to 64 teams, the number of teams advancing to the UCL.

[data]





**Conclusion: Through clustering, we matched 61 of the 64 teams advancing to the UCL, excluding the above 3 teams.**

II-3) To find criteria of not being relegated (by classification) + Conclusion

|  |
| --- |
| in [1]  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.preprocessing import OrdinalEncoder  from sklearn.model\_selection import train\_test\_split, GridSearchCV  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  from pandas import DataFrame  import warnings  warnings.filterwarnings("ignore") |

|  |
| --- |
| in [2]  #attach csv file  EPL = pd.read\_csv('ranking.csv')  EPLEPL = pd.read\_csv('ranking.csv')  EPL  #YEAR: Year of league  #RANK: Ranking  #GF: Total goals for  #GA: Total goals against  #GD: Total goals difference  #WIN: Total number of matches won  #DRAW: Total number of matches drawn  #LOSE: Total number of matches lost  #POINT:  #EPL: Yes or no to win EPL  #UCL: Yes or no to get a ticket to UCL  #RELEGATION: Yes or no to be relegated |

|  |
| --- |
| out [2]    320 rows × 12 columns |

|  |
| --- |
| in[3]  # data preprocessing  # Because I set GF,GA, WIN, DRAW, LOSE as classifying attribute, and RELEGATION as target attribute, I will delete attributes 'YEAR','RANK','GD','POINT','EPL', 'UCL' which are not related to this decision tree.  EPL1=EPL.drop(['YEAR','RANK','GD','POINT','EPL', 'UCL'],axis=1,inplace=True) |

|  |
| --- |
| in[4]  # After data preprocessing  EPL1 |

|  |
| --- |
| out [4]  GF GA WIN DRAW LOSE RELEGATION  0 73 35 25 7 6 no  1 66 37 24 6 8 no  2 70 34 20 11 7 no  3 52 35 18 9 11 no  4 49 32 17 12 9 no  ... ... ... ... ... ... ...  315 40 61 9 15 14 no  316 46 66 11 7 20 no  317 37 58 8 15 15 yes  318 55 78 10 9 19 yes  319 43 70 7 12 19 yes |

|  |
| --- |
| in [5]  #for classification, I set training data : test data = 7:3  feature\_cols = ['GF', 'GA', 'WIN', 'DRAW', 'LOSE']  X = EPL[feature\_cols]  y = EPL.RELEGATION  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=100) |

|  |
| --- |
| in [6]  print("Shape of X\_train: {}".format(X\_train.shape))  print("Shape of y\_train: {}".format(y\_train.shape))  print("Shape of X\_test: {}".format(X\_test.shape))  print("Shape of y\_test: {}".format(y\_test.shape)) |

|  |
| --- |
| out [6]  Shape of X\_train: (224, 5)  Shape of y\_train: (224,)  Shape of X\_test: (96, 5)  Shape of y\_test: (96,) |

|  |
| --- |
| in [7]  #Now, I will make decision tree by using training data and test data.  # To use decision tree, I have to decide criterion ( ex)gini, entropy ) and max depth of decision tree. |

|  |
| --- |
| in [8]  param\_grid = {  'criterion':['gini', 'entropy'],  'max\_depth': list(range(2, 16)),  'min\_samples\_leaf': list(range(1, 6)),  'min\_samples\_split': list(range(2, 6)),  'random\_state':[10]  }  clf = GridSearchCV(DecisionTreeClassifier(), param\_grid, n\_jobs=-1, cv=5) |

|  |
| --- |
| in [9]  # I have to train model.  %time clf.fit(X\_train, y\_train) |

|  |
| --- |
| in [10]  clf.best\_params\_ |

|  |
| --- |
| out [10]  {'criterion': 'gini',  'max\_depth': 2,  'min\_samples\_leaf': 1,  'min\_samples\_split': 2,  'random\_state': 10}  #Result shows that proper criterion to divide is gini and proper max\_depth is 2. |

|  |
| --- |
| in [11]  # To make decision tree by classification, I have to check accuracy.  my\_model = clf.best\_estimator\_  my\_model.fit(X\_train, y\_train)  my\_model\_score\_train = my\_model.score(X\_train, y\_train)  my\_model\_score\_test = my\_model.score(X\_test, y\_test)  print(clf.best\_estimator\_)  print('Training data의 accuracy = ',my\_model\_score\_train)  print('Testing data의 accuracy = ',my\_model\_score\_test) |

|  |
| --- |
| out [11]  DecisionTreeClassifier(max\_depth=2, random\_state=10)  Training data의 accuracy = 0.9508928571428571  Testing data의 accuracy = 0.9791666666666666  # Enough accuracy |

|  |
| --- |
| in [12]  # To make decision tree, I have to find importance of every attributes. Dividing proceed by importance of attributes.  features = pd.DataFrame(X.columns.tolist())  features.columns = ['Features']  fi = pd.DataFrame(my\_model.feature\_importances\_)  fi.columns = ['Importance']  fea\_imp = pd.concat([features, fi], axis=1)  fea\_imp.sort\_values(by='Value', ascending = False) |

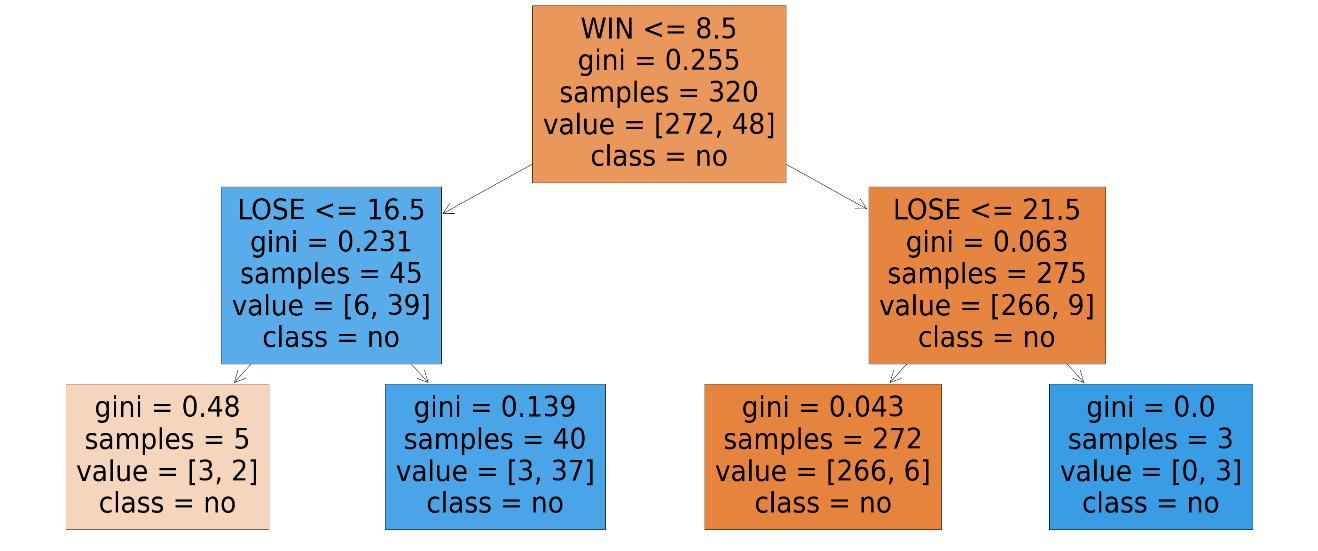
|  |
| --- |
| out [12]  Features Importance  2 WIN 0.80419  4 LOSE 0.19581  0 GF 0.00000  1 GA 0.00000  3 DRAW 0.00000  # I will divide tree by this order. Max depth is 6.  So maybe attributes ‘WIN’ and ‘LOSE’ are used. |

|  |
| --- |
| in [13]  # It’s time to make decision tree.  from sklearn.tree import DecisionTreeClassifier  treeclf = DecisionTreeClassifier(max\_depth=3, random\_state=1)  treeclf.fit(X, y)  from sklearn import tree  text\_representation = tree.export\_text(treeclf)  print(text\_representation)  #feature\_0: Total goals for(GF)  #feature\_1: Total goals against(GA)  #feature\_2: Total number of matches won (WIN)  #feature\_3: Total number of matches drawn (DRAW)  #feature\_4: Total number of matches lost (LOSE) |

|  |
| --- |
| out [13]  |--- feature\_2 <= 8.50  | |--- feature\_4 <= 16.50  | | |--- class: no  | |--- feature\_4 > 16.50  | | |--- class: yes  |--- feature\_2 > 8.50  | |--- feature\_4 <= 21.50  | | |--- class: no  | |--- feature\_4 > 21.50  | | |--- class: yes  # It’s just a frame of decision tree, not complete one. |

|  |
| --- |
| in [14]  #Now it’s time to make complete decision tree.  with open("decision\_tree.log", "w") as fout:  fout.write(text\_representation)  fig = plt.figure(figsize=(50,20))  \_ = tree.plot\_tree(treeclf,  feature\_names=feature\_cols,  class\_names=EPL.RELEGATION.astype(str),  filled=True) |

out [14]



Conclusion (not to be relegated)

Among 38 matches,

1. WIN <=8.5, LOSE <=16.5 – no relegation

2. WIN <=8.5, LOSE >16.5 – relegation

3. WIN > 8.5, LOSE <=21.5 – no relegation

4. WIN > 8.5, LOSE > 21.5 – relegation

**Do Association Rule for relegation +Conclusion**

[1]

import pandas as pd

import numpy as np

from pandas import DataFrame

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

First, import the libraries

[2]

EPL = pd.read\_csv('Downloads/ranking.csv')

[3]

EPL.head()

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

[4]

EPL1=EPL.drop(['YEAR','RANK','GD','POINT','EPL','UCL'],axis=1,inplace=True)

Drop the columns that are not relative

[5] EPL.head()

테이블이(가) 표시된 사진

자동 생성된 설명

[6]

epl\_copy = EPL.copy() #copy the dataset

[7]

epl\_copy ['GF']=pd.qcut(epl\_copy['GF'],q=4,labels=['gfq1','gfq2','gfq3','gfq4'])

epl\_copy['GA']=pd.qcut(epl\_copy['GA'],q=4,labels=['gaq1','gaq2','gaq3','gaq4'])

epl\_copy['WIN']=pd.qcut(epl\_copy['WIN'],q=4,labels=['winq1','winq2','winq3','winq4'])

epl\_copy['DRAW']=pd.qcut(epl\_copy['DRAW'],q=4,labels=['drawq1','drawq2','drawq3','drawq4'])

epl\_copy['LOSE']=pd.qcut(epl\_copy['LOSE'],q=4,labels=['loseq1','loseq2','loseq3','loseq4'])

Divide the quartile for each columns

[8]

eplvalue = epl\_copy.values.tolist()

eplvalue

테이블이(가) 표시된 사진

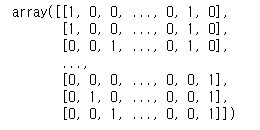
자동 생성된 설명

[9]

te = TransactionEncoder()

te\_ary = te.fit(eplvalue).transform(eplvalue)

te\_ary.astype('int')



Encode for association rule

[10]

df = pd.DataFrame(te\_ary, columns=te.columns\_)

frequent\_itemsets = apriori(df, min\_support=0.1, use\_colnames=True)

frequent\_itemsets

테이블이(가) 표시된 사진

자동 생성된 설명

[11]

rules = association\_rules(frequent\_itemsets, metric='support', min\_threshold = 0.1)

rules[rules['consequents'] == {'yes'}]

테이블이(가) 표시된 사진

자동 생성된 설명

**Conclusion**

**we found that highly relevant to relation is win, lose, goal against and second is lose, win.**

**It is same with tree that win and lose is important factor.**

**Role of Members**

Kwak Yu Seok : Data collecting, Finding decision tree by classification

Kim Wu Yeol : PPT, Final revision

Kim Jeong Yong : Clustering

Choi Ji Woong : Data Processing, Association Rule