For office use only	Team Control Number	For office use only
T1 _____	**2013647**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**D**	F4 _____

2020
MCM/ICM
Summary Sheet

Team Evaluation Using Network Science and Classifier

Summary

In order to provide effective team cooperation mode and strategies for Huskies' soccer coach for future improvement, we built passing networks combined with machine learning models based on comprehensive analysis on the data given.

For task 1, we firstly did preliminary analysis on the correlation of the location of passing the ball and the outcome of the match. The results turned out that there was little correlation between the two. Hence, secondly, we created the networks from multiple-scales analysis. We computed different network metrics, which represented different meanings of centrality, and tried to find correlation between these metrics and the outcome of the match.

For task 2, we created the Activity Level Frequency Model based on the network analysis, which could represent the frequency a team made of certain actions in one match. In addition, we did Fourier transformation to this model to calculate how many effective actions a team made in one match. Further, we utilized machine learning techniques to fit our models on the data. By evaluating our machine learning models, we evaluated how accurate our metrics describe a soccer team.

For task 3, we summarized the prediction results using our models, based on which, we offered specific suggestions to Huskies coach for promoting the team's performance next season.

For task 4, based on the analysis of the Huskies team, we extended the setting of teamwork to complex social backgrounds, and proposed the understanding of Group Dynamics, how to design a high-quality team, and additional significant indicators that affect teamwork.

Keywords: Network Science; Network Metrics; Team Evaluation; Random Forest; Fourier Transformation; Group Dynamics

Contents

1 Introduction

1.1 Background

With the increasing demand of people in today's society to rely on teamwork to solve increasingly complex social problems in all aspects, our pursuit and exploration of the factors that contribute to team success have become significant and indispensable. People are dedicated to continuous research on optimizing strategies for achieving higher level of teamwork, and competitive team sports is one of the biggest areas that could benefit from these studies.

How can we give full play to our team's initiative and break the opponent team's strategy to win under the strict rules of the game? In order to solve this problem, we not only need to analyze the skill levels and kicking modes of individual players, tactical strategy and internal cooperation of the whole team, individual player's different influence on the team are also important. It is worth noting that the above analysis needs to be carried out under various settings, including own team, opponent teams, different games, different zooms in the playing field and different time periods of the game, so as to achieve a comprehensive, detailed and accurate team analysis and improvement effect.

1.2 Restatement of the Problem

According to the Huskies coach's requests, we need to use Huskies' game data in last season to quantify and regularize the team's structure and dynamic characteristics. Specifically, analyzing how the complex interaction between the players on the field affects the team's scoring, the team's game state and vitality, and puts forward specific improvement suggestions on teamwork for the training and games of Huskies next season based on the analysis.

For Task 1, firstly, we are supposed to build passing networks for the ball passing between players, with players being nodes and passes being links, and then process, model and analyze the passing process of each game to identify network patterns. Secondly,

For Task 2, determine the specific indicators that affect the quality of teamwork, and build a model from the team level through the combination of these indicators to reflect the structural, configurational, and dynamical aspects of teamwork.

For Task 3, based on the analysis of our team network and model, point out the effective team cooperation mode and strategies for Huskies, and specifically suggest the aspects that they need to improve in the next season.

For Task 4, reflect on our analysis on Huskies, considering the indicators and factors that improve team performance, we are supposed to generalize how to design and build a higher-quality team and point out additional team cooper-

ation indicators needed to analyze a team.

1.3 Overview of Our Work

We firstly did basic analysis on the data given and calculated the correlation with the output. Secondly, we used graph-tool which is a library in C++ to create the network for each match. After that, we computed different network metrics and their correlation with the outcome and the score for each match. Then we created the Activity Level Frequency Model, finding significant results from multiple scales and visualizing the data in a better way. Next, we used some methods of machine learning such as using classifiers like random forest to get the inner visualization of the network metrics and the outcome of the matches.

2 Analysis of the Problem

2.1 Correlation Analysis and Heat Map

According to the data given, we computed the correlation among the passing position, event type, event sub type and the outcome. Here we use the ϕ coefficient of chi-squares test, where $\phi = \sqrt{\frac{\mathcal{X}^2}{n}}$. The advantage of using ϕ coefficient is it can test the association between several variables and is always bigger than zero. The following is the result.

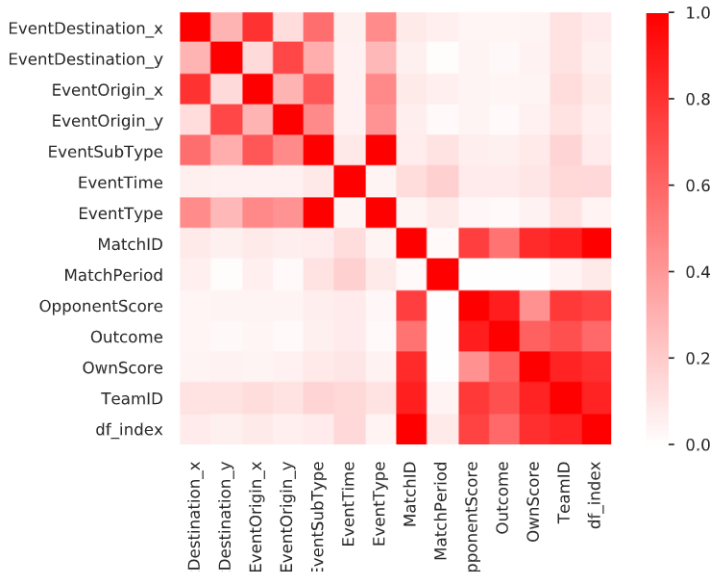

Figure 1: ϕ_k coefficient between different variables.

We can see that the area of passing position and outcome has very weak association.

2.2 Network Creation

We create nodes to represent each player, and add edges between two nodes to represent that they passed one ball. Then we created the network as the following:

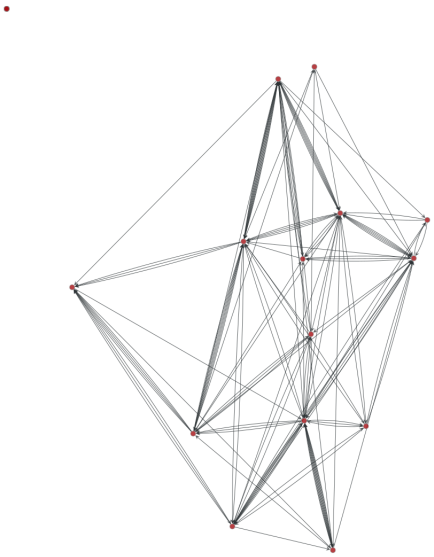

Figure 2: Network example plot of the 12^{th} match.

The thickness of the edges represents the number of passes between two nodes.

2.3 Network Analysis

We calculated different metrics to find the correlation with the outcome.

1. Local Clustering Coefficient[4]
   Our network is a directed graph, here the local clustering coefficient is defined by:
   $$C_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i\,(k_i - 1)}$$
   Where N_i represents the set of the neighbourhoods of vertex i, E represents the set of all the edges in this network, k_i represents the number of the neighbourhoods of vertex i. Then we calculate the mean value, standard deviation of C_i, $i \in [0, n]$, and get the correlation as the following:

	mean	std	Outcome	OwnScore	ScoreDiff
mean	1.000000	0.631727	0.185350	0.229202	-0.103709
std	0.631727	1.000000	-0.005841	0.189648	-0.266331
Outcome	0.185350	-0.005841	1.000000	0.695860	0.180965
OwnScore	0.229202	0.189648	0.695860	1.000000	-0.580393
ScoreDiff	-0.103709	-0.266331	0.180965	-0.580393	1.000000

Table 1: Result of correlation of clustering coefficient.

2. Closeness Centrality[4]
   Here the closeness centrality can be defined by:

$$c_i = \frac{1}{\sum_j d_{ij}}$$

   This metric can represent how tight a network is. The analysis of correlation is as the following:

	mean	std	Outcome	OwnScore	ScoreDiff
mean	1.000000	0.066782	-0.001622	0.102358	-0.146188
std	0.066782	1.000000	0.107140	0.044999	0.058649
Outcome	-0.001622	0.107140	1.000000	0.695860	0.180965
OwnScore	0.102358	0.044999	0.695860	1.000000	-0.580393
ScoreDiff	-0.146188	0.058649	0.180965	-0.580393	1.000000

Table 2: Result of correlation of closeness centrality.

3. Betweeness Centrality[4]
   Here the betweeness centrality can be defined by:

$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq v}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

   Here σ_{st} is the number of shortest paths from s to t. $\sigma_{st}(v)$ represents the number of shortest paths from s to t that pass through a vertex v. This mertric can represents organization of midfield formation. The analysis of correlation is as the following:

4. Eigenvector Centrality[4]
   Here the eigenvector centrality can be defined by:

$$Ax = \lambda x$$

	mean	std	cpd	Outcome	OwnScore	ScoreDiff
mean	1.000000	0.489426	0.264283	0.124010	-0.092904	0.267839
std	0.489426	1.000000	0.882390	0.073674	0.028949	0.043895
cpd	0.264283	0.882390	1.000000	-0.000588	0.028128	-0.039186
Outcome	0.124010	0.073674	-0.000588	1.000000	0.695860	0.180965
OwnScore	-0.092904	0.028949	0.028128	0.695860	1.000000	-0.580393
ScoreDiff	0.267839	0.043895	-0.039186	0.180965	-0.580393	1.000000

Table 3: Result of correlation of betweeness centrality.

Where λ is the largest eigenvalue of adjacency matrix A. Adjacency matrix for directed network is defined by:

$$A_{ij} = \begin{cases} 1 & \text{if } (j,i) \in E \\ 0 & \text{otherwise} \end{cases}$$

A_{ij} corresponds to the directed edge $j \to i$. The v^{th} component of the related eigenvector then gives the relative centrality score of the vertex v in the network. The correlation analysis is as the following:

	eigenvector	Outcome	OwnScore	ScoreDiff
eigenvector	1.000000	0.156880	0.195184	-0.089405
Outcome	0.156880	1.000000	0.695860	0.180965
OwnScore	0.195184	0.695860	1.000000	-0.580393
ScoreDiff	-0.089405	0.180965	-0.580393	1.000000

Table 4: Result of correlation of eigenvector centrality.

5. Katz Centrality[4]
   Here Katz centrality can be defined by:

$$\mathbf{x} = \alpha \mathbf{A}\mathbf{x} + \beta$$

Where $\mathbf{A}$ is the adjacency matrix defined above, and β is the personalization vector. Here we assume $\beta = 1$. The correlation analysis is as the following:

3 Activity Level Frequency Model Analysis

3.1 Model Creation

From the analysis given above, we did not find obvious correlation between centrality metrics and the match outcome.

	mean	std	Outcome	OwnScore	ScoreDiff
mean	1.000000	-0.488159	-0.054294	-0.220602	0.240535
std	-0.488159	1.000000	0.189791	0.263458	-0.145584
Outcome	-0.054294	0.189791	1.000000	0.695860	0.180965
OwnScore	-0.220602	0.263458	0.695860	1.000000	-0.580393
ScoreDiff	0.240535	-0.145584	0.180965	-0.580393	1.000000

Table 5: Result of correlation of Katz centrality.

In this model, we define every event happened in the match will contribute to the four parts: attack, defense, collaborate, foul. And each subevent will contribute different weights to these four parts. We cut the whole match into many small time intervals . Let $S \in \mathbb{R}^{m\times 4}$ to denote the final activity level of the team at the end of the match.

$$S = \sum_{i=1}^{n} \left(e_i \cdot \sum_{p_i=1}^{q_i} \alpha_i \right)$$

Where n is the number of time intervals, q_i is the number of events happening in the $i^{\mathbf{th}}$ time interval, $\alpha_i^{\mathbf{T}} \in \mathbb{R}^4$ denotes to the different weights one event contribute to attack, defense, collaborate and foul. $e_i \in \mathbb{R}^m$ which $i^{\mathbf{th}}$ component is 1 and 0 otherwise.

The four columns of S represent the activity level of the whole team's attack, defense, collaborate, foul respectively. The list of α_i is in the appendix. We get the result as the following:

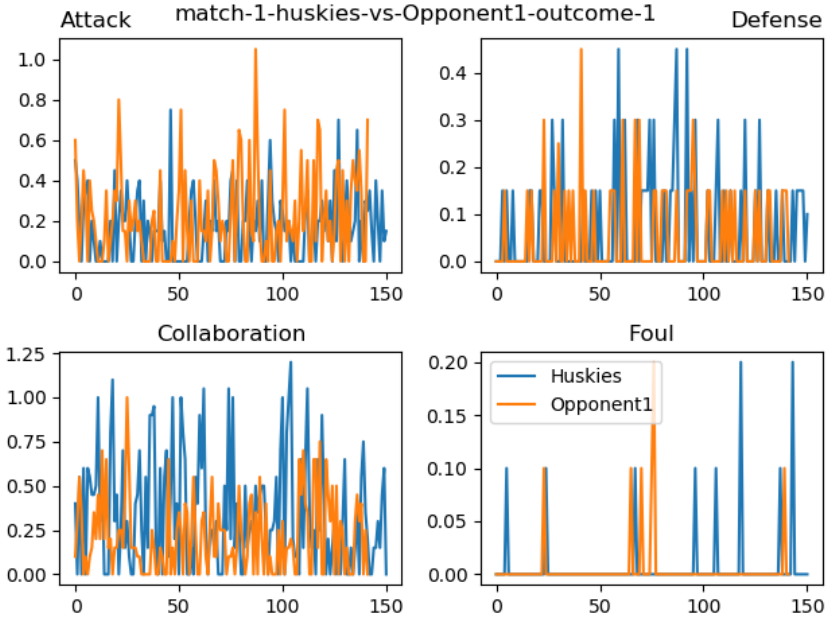

Figure 3: Activity Level of one winning match.

We can clearly see that Huskies' collaboration frequency level is larger than Opponent 1.

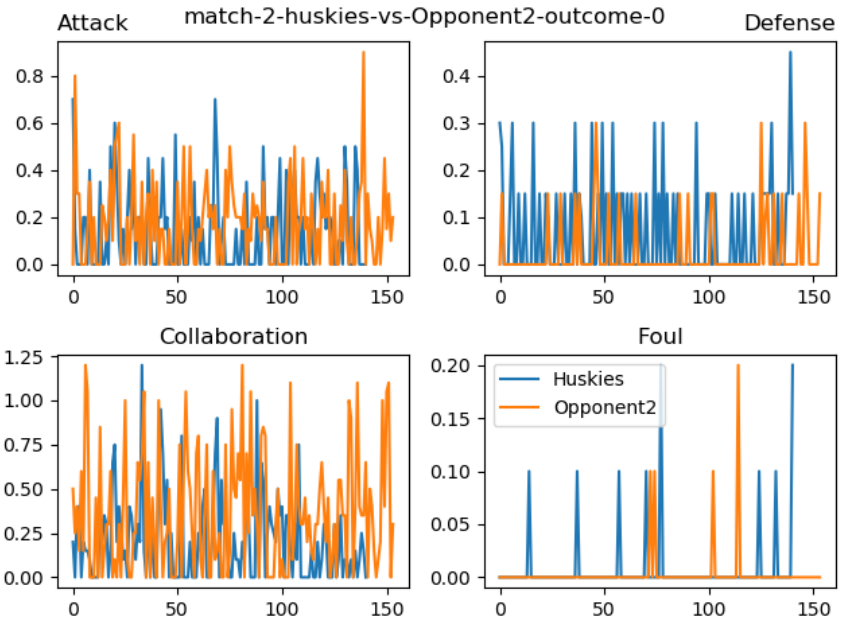

Figure 4: Activity Level of one tie match.

We can see that although Huskies' collaboration is far worse then Opponent2's, but our defense is really good, so it is a tie match.

3.2 Fourier Transformation of the Frequency Model

Since we've already built the Activity Level Model and got the frequency, it is still not so obvious to find out. So we did Fourier transformation to the activity frequency level model. After the transformation, we can find the response to the high frequency, which means that we can find out how many times one team can attack, defense, collaboration or foul at a certain frequency, the number of effective attacking, collaboration etc. The result is as the following:

Since the Fourier transformation result is symmetric. So the middle part of each plot represents the response of the highest frequency. Looking at the collaboration part, Huskies' number of high frequency collaboration is much higher than Opponent1.

Here's one example of a losing match.

In this figure, we can see that Opponent4's effective collaboration is much higher than Huskies', and their effective attack is also a little bit higher than Huskies'.

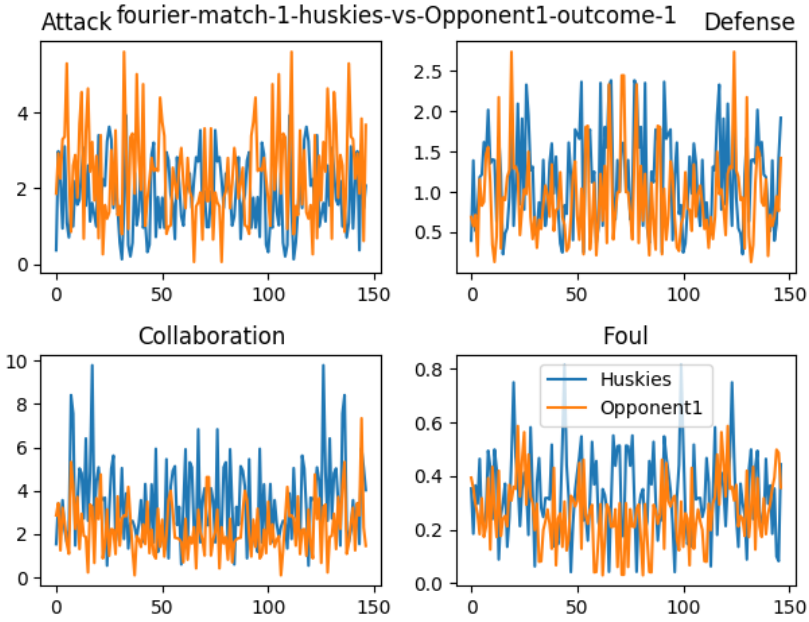

Figure 5: Fourier Transformation of the first match's frequency model.

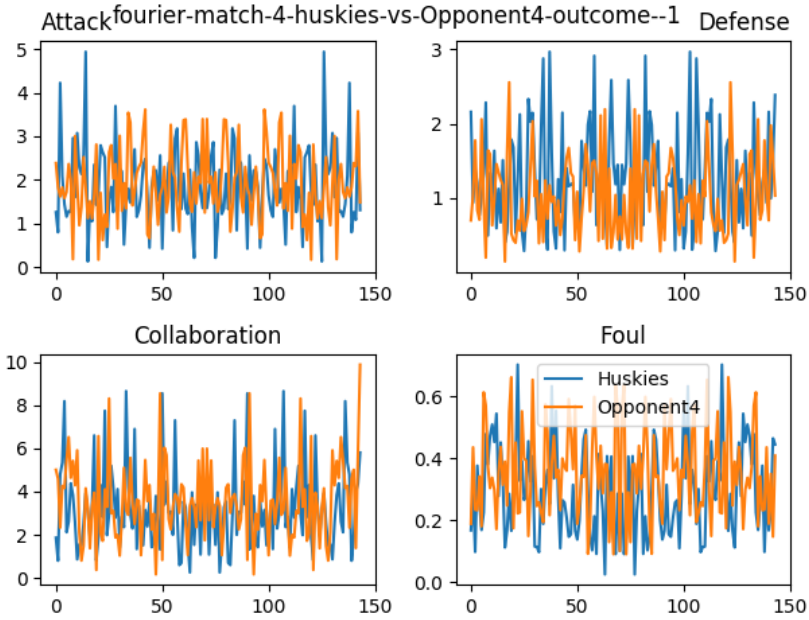

Figure 6: Fourier Transformation of the 4^{th} match's frequency model.

4 Model Implementations Using Machine Learning

4.1 Goals

Our intended goals to achieve with the help of machine learning techniques are:

- generalizing the relationship of some indicators with the overall performance of a soccer team[2]
- being able to make prediction of a team's performance in the match with the some input indicators
- being easy to extended to a more general model that can fit many other types of social groups, such as political groups, companies, and nations, as well as being able to provide insights on how to improve their performance

To be more specific, our model must at least be able to use the network metrics and activity level frequencies introduced above to predict the outcome of a match.

4.2 Model Selection

There are plenty of models, widely used in the realm of data science, to choose from. However, there are several crucial factors that affects the quality of the final model.

- the size of the data
- the type of the data (categorical, numerical, etc.)
- the evaluation (score) methods
- the optimization methods

4.2.1 Data Size

Since the dataset is very small (extremely small compared to datasets that many machine learning projects used), it is a good idea to avoid classifiers that requires a large amount of data in order to manage to reach a satisfying accuracy. Moreover, the probability of over-fitting appearing is small, so regularization is not likely to be needed.

4.2.2 Data Type

For data types, we are mainly concerned with the target (output) type, since the inputs are all numerical data (clustering coefficients, centrality coefficients, etc.) However, we mustn't ignore the fact that some inputs are matrices while the other are scalar values. We will address this later in this paper in more details.

In the current context, our output data consists of categorical data (match outcomes). Therefore, classification models are required to make the prediction.

4.3 Random Forest

For the classifier model, we chose to use Random Forest to fit the data

4.3.1 Algorithms

The process of the algorithm of random forest is as the following:

1. Randomly pick a random sample with replacement of the training set, and repeated this process for $m \times n$ times, in order to split into m sub parts, each sub part has n samples, which is same with the number of original training set. This step is called bagging step.
2. Created decision trees for each sub set.
3. Averaging the predictions from all the individual regression trees.

4.3.2 Feature Extraction

Two sets of feature are extracted in this process, one is the metrics discussed in 2.3 Network Analysis and the other one is the metrics introduced in 3 Activity Level Frequency Model Analysis.

For the network metrics, seven of them are included in the training data, they are Passing Volume, Clustering Coefficient, Pagerank Centrality, Closeness Centrality, Betweenness Centrality, Eigenvector Centrality and Katz centrality. Since every player in the network have their own coefficient[1], the resulting coefficients of a network is a list of individual types of coefficients. To reduce the dimension of the data, we simply concatenate all metrics into a list. Therefore, we use one single list of values to represent the "feature" of a network (or a match).

For the activity frequency metric, we follows similar procedures as above. Four types of scores (attack, defense, collaboration, and fouls) are concatenated along timeline and thereby the resulting feature of a match is a list of values.

4.3.3 Training

Two separate models are created and trained using the sets of feature. Then each of them is validated using K-Fold Cross Validation.

4.4 Tuning

The followings are important hyper-parameters for the model:

- max depth: the max depth of the trees
- max features: the number of features to consider when looking for the best split
- number of estimators: the number of trees (estimators) in the forest

Using a hyper-parameter tuning technique called Grid Search, we can find out the best hyper-parameter for the model on this set of data.

Grid Search is an exhaustive searching of the best parameters through a specified hyper-parameter subspace. In other words, Grid Search can help us find the hyper-parameters that make the model perform (almost) the best on a certain dataset. However, since the searching range is specified manually, it can miss the global best parameter(s).

The hyper-parameters for the network-based model are:

- max depth $= 2$
- max features $\log_2(\text{number of features})$
- number of estimators $= 80$

The hyper-parameters for the activity-based model are:

- max depth $= 3$
- max features $\sqrt{\text{number of features}}$
- number of estimators $= 90$

5 Model Evaluation

The network-based model can reach a maximum $\mathbf{R}^2$ score of 0.75 and minimum of 0.25.

The activity-based model can reach a maximum $\mathbf{R}^2$ score of 0.675 and minimum of 0.125.

Obviously, the model quality varies greatly throughout different training pass. However, considering the small size of the data, it can clearly show the implication of the two types of metrics. Also, since Random Forest rely largely on random processes, the scores of the model varies greatly each time it is fitted. Hence, it is difficult to develop a comprehensive and accurate evaluation of the models.

6 Suggestions for Huskies

- Promote high frequency collaboration
- Increasing the number of maintaining high frequency of attack, collaboration.
- Use positional strategies like 4-4-3 to tighten the formation and decrease the distance for players to reach others

7 Generalizations in Effective Teamwork

7.1 Group Dynamics

By modeling and analyzing the performance of Huskies' team in last season, we found that player's individual capabilities, the complexity of the team's attack, defense, collaborate, foul modes and frequencies all have important effects on team dynamics. We extend the soccer problem to more diverse and complex teamwork issues in society, summarize the multi-dimensional factors that reflect and affect group dynamics, and use the following mind map to represent:

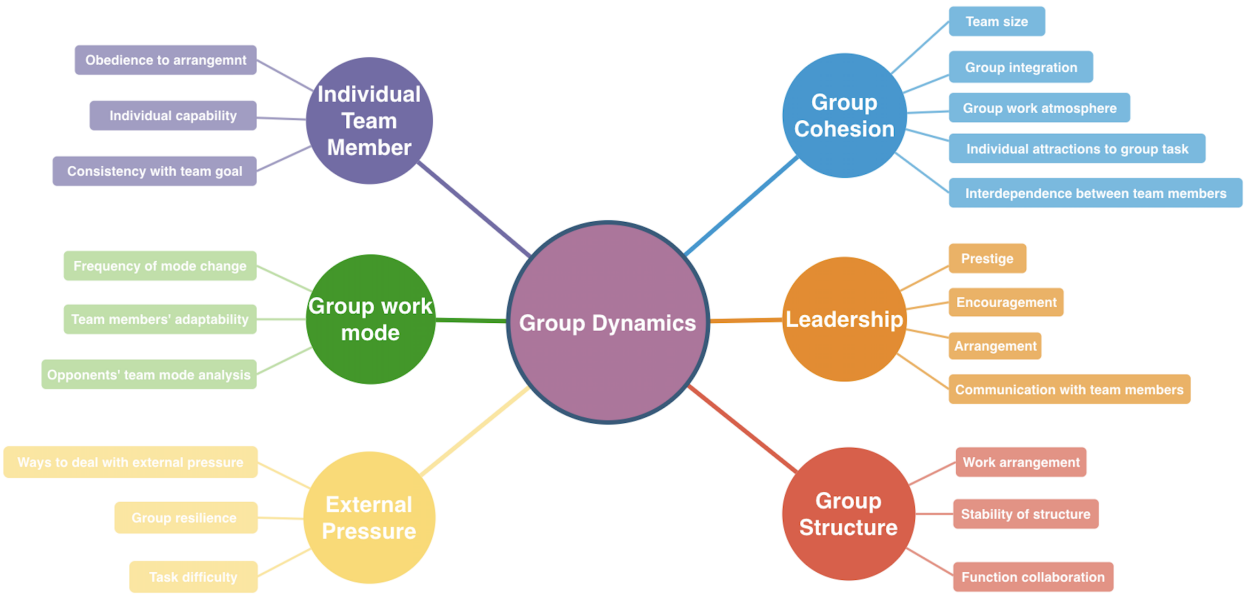

Figure 7. Mind Map of Group Dynamics.

7.2 Implicational Generalizations in Team Design

7.2.1 Team Standard

We should set criteria for selection of leaders and team members: In the case of clear team goals, it is necessary to formulate standards in order to select the appropriate team members and team leaders. This not only helps the formation of high-quality teams, but also updates the standards to give the members appropriate pressure and promote the continuous progress and development of team members . For instance, the marginal players in Huskies team will constantly strive to improve their strength to win more playing time and contribute more, and the core members will also keep fighting in order to maintain their core position. In this process, the overall strength of the team would be improved to a large degree

7.2.2 Team Cohesion

Team cohesion: a dynamic process that reflects the tendency of a group to remain united to pursue its instrumental goals and/or to meet the effective needs of its members[3]. Researches show that team cohesion has a positive relationship with the team's success in sports, and it is universal to social teamwork. Therefore, when designing an effective team, we need to ensure that this team has high team cohesion. This could be achieved by the following ways:

- Increase the interdependence between team members: Huskies players work together, cooperate with each other, and give their peers full trust. In complex social teamwork, if team members are highly integrated with other members in behavior, emotion and psychology, it would easy to form a joint force, thus raising team cohesion; on the contrary, if the degree of interdependence in the achievement of goals is low, it would be hard to form team cohesion.
- Establish a team incentive model and promote a positive team atmosphere: under such model and atmosphere, team members can maintain their own good inner emotions, and at the same time, they can encourage other team members from different levels. The team 's overall morale and confidence are strengthened to better complete tasks when members are supported by more psychological power.
- Form appropriate teamwork structure: When a team has a stable team structure, the cooperation between players will be more effective, and this dynamic balance can be maintained continuously. Therefore, in teamwork, appropriate and stable structures need to be designed for the responsibilities and arrangements of team members need to according to different settings, which can improve team cooperation, thus promoting the formation of stable team cohesion.

7.2.3 Leadership

In a football team, the coach and the team captain are the two leaders. The former trains the players, formulates the game strategy, field instructions and post-match summary; the latter's performance in the game will directly affect the team's performance. However, both of them need to establish prestige among team members, and play a role in encouraging morale and promoting team cohesion. From this point of view, the existence of leadership has great significance in social teamwork. In order to create a high-quality team, it is necessary to choose the right person for the leader, make them fully clear their responsibilities, mobilize the morale of the team, and direct the work of the leadership team.

7.2.4 Individual Team Member

- Clarify the division of labor and responsibilities of each member: The goalkeeper, forward, midfielder and backcourt in Huskies have their most active areas and designated tasks. Only players in different positions play their strengths, complete their duties and actively cooperate with other members in the team can they score as many points as possible. Thus, under complex settings of teamwork, each member of the team needs to have a clear understanding of their own strengths and weaknesses, as well as other members of the team. On this basis, identify their main tasks and try their best to complete them.
- Obey instructions and arrangements: Players must obey the overall strategic arrangement of the coach during the game. They cannot completely go their own way and forget the team. Therefore, in teamwork, each team member must follow the overall arrangement of the team, and only the obedience of the team members can promote the orderly cooperation of a team.

7.3 Additional Influencing Factors

In order to to develop a more comprehensive generalized model of team performance, we also need detailed data on the following indicators, which might not be used in analyzing soccer game:

- Team size: the size of the team is likely to affect the cohesion of the team under the different settings of team cooperation, (the larger the team size, the less opportunities and possibilities for interaction between team members, which makes it difficult to form cohesion; on the contrary, if reducing team size, too much interaction may cause unnecessary contradictions and affect cohesion). Therefore, we need detailed data to analyze the impact of team size on team cohesion under specific settings.

- Task difficulty: the difficulty of a task directly affects team members' confidence and expectations for completing the task, and then affects team cohesion. By capturing the impact of task difficulty on team cohesion, we can consider adjusting the task difficulty in different ways in teamwork to improve the success rate of each task completion.
- Consistency of team members' goals: whether members within the team have consistent goals has an significant impact on achieving effective teamwork. Self-consciousness within the members will not only affect team cohesion, but also directly affect the team's ultimate success.
- External pressure level: threats from outside sources to force the team to accomplish the goal. Our preliminary assumption is that too high or too low external pressure is not conducive to promoting team vitality, so we need to find an appropriate level, and the definition of "appropriate" here depends on relevant index data.

8 Strengths and Weaknesses

8.1 Strengths

- It is easy to visualize the performance in four aspects - attack, defense, collaboration and foul of one team in a match from our model.
- After evaluation, despite the small amount of data, the accuracy of our model is still relatively high.
- We did analysis on time-series activity data, which is an innovative idea.
- Fully utilize all aspects of data, for instance, our analysis covered both starting and alternate players

8.2 Weaknesses

- Possible to miss some detailed information hidden behind the frequency, such as some important events but happened only once.
- The current model is preliminary quantified, a larger size of data is needed for further improvement.

References

[1] Buldú, J.M., Busquets, J., Echegoyen, I. et al. Defining a historic football team: Using Network Science to analyze Guardiola's F.C. Barcelona. *Sci Rep*, no. 9, (2019)

[2] Cintia, P., Giannotti, F., Pappalardo, L., Pedreschi, D., & Malvaldi, M. The harsh rule of the goals: Data-driven performance indicators for football teams. *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, (2015): 1-10.

[3] Carron, A.V., Brawley, L.R. and Widmeyer, W.N. Measurement of cohesion in sport and exercise. *In Advances in Sport and Exercise Psychology Measurement*, (1998): 213-226.

[4] `https://graph-tool.skewed.de/static/doc/centrality.html`

Appendices

data.py

```
import numpy as np
import pandas as pd

__all__ = [
    'matches_df',
    'passings_df',
    'events_df',
    'match_ids',
    'huskies_passes',
    'huskies_events',
    'huskies_player_ids',
    'opponent_player_ids',
    'all_events',
]

data_matches_path = '../data/matches.csv'
data_passings_path = '../data/passingevents.csv'
data_events_path = '../data/fullevents.csv'

matches_df = pd.read_csv(data_matches_path)
passings_df = pd.read_csv(data_passings_path)
events_df = pd.read_csv(data_events_path)


def outcome_int_map(x: str):
    if x == 'win':
        return 1
    elif x == 'tie':
        return 0
    else:
        return -1


matches_df['Outcome'] = matches_df['Outcome'].apply(outcome_int_map)

all_events = events_df.join(
    matches_df[['OwnScore', 'OpponentScore', 'Outcome']],
    on='MatchID',
    how='outer')

huskies_passes = all_events[(all_events['TeamID'] == 'Huskies')
                            & (all_events['EventType'] == 'Pass')]
huskies_events = all_events[all_events['TeamID'] == 'Huskies']

huskies_player_ids = ['Huskies_D1', 'Huskies_D10',
                      'Huskies_D2', 'Huskies_D3',
                      'Huskies_D4', 'Huskies_D5',
                      'Huskies_D6', 'Huskies_D7',
                        'Huskies_D8', 'Huskies_D9',
                        'Huskies_F1', 'Huskies_F2',
```

```
                    'Huskies_F3', 'Huskies_F4',
                    'Huskies_F5', 'Huskies_F6',
                    'Huskies_G1', 'Huskies_M1',
                    'Huskies_M10', 'Huskies_M11',
                    'Huskies_M12', 'Huskies_M13',
                    'Huskies_M2', 'Huskies_M3',
                    'Huskies_M4', 'Huskies_M5',
                    'Huskies_M6', 'Huskies_M7',
                    'Huskies_M8', 'Huskies_M9']

match_ids = np.unique(matches_df['MatchID'])
```

activity_level.py

```
from soccer_network.data import *
from typing import Tuple, List
from matplotlib import pyplot as plt
import matplotlib as mpl
import pandas as pd
import numpy as np

mpl.rc("savefig", dpi=200)

__all__ = ['get_activity_level']

activity_scores = {
    'Duel': {'Air duel': (0, 1.0, 0, 0),
             'Ground attacking duel': (.15, 0, 0, 0),
             'Ground defending duel': (0, .15, 0, 0),
             'Ground loose ball duel': (.1, 0, 0, 0)
             },
    'Foul': {'Foul': (0, 0, 0, .1),
             'Hand foul': (0, 0, 0, .3),
             'Late card foul': (0, 0, 0, .1),
             'Out of game foul': (0, 0, 0, .3),
             'Protest': (0, 0, 0, 1.),
             'Simulation': (0, 0, 0, .1),
             'Time lost foul': (0, 0, 0, .1),
             'Violent Foul': (.05, 0, 0, .15)
             },
    'Free Kick': {'Corner': (.1, 0, 0, 0),
                  'Free Kick': (.1, 0, 0, 0),
                  'Free kick cross': (.1, 0, .1, 0),
                  'Free kick shot': (.15, 0, 0, 0),
                  'Goal kick': (.2, 0, 0, 0),
                  'Penalty': (.1, 0, 0, 0),
                  'Throw in': (.2, 0, 0, 0)
                  },
    'Goalkeeper leaving line': {'Goalkeeper leaving line':
        (0, 0, 0, .2)},
    'Interruption': {'Ball out of the field': (0, 0, 0, 0),
    'Whistle': (0, 0, 0, 0)},
    'Offside': (0, 0, 0, .1),
    'Others on the ball': {'Acceleration': (0, 0, 0, 0),
```

```
                            'Clearance': (0, .15, 0, 0),
                            'Touch': (0, 0, .15, 0),
                            },
    'Pass': {'Cross': (0, 0, .15, 0),
             'Hand pass': (0, 0, .1, 0),
             'Head pass': (0, 0, .15, 0),
             'High pass': (0, 0, .15, 0),
             'Launch': (0, 0, .1, 0),
             'Simple pass': (0, 0, .1, 0),
             'Smart pass': (0, 0, .2, 0)
             },
    'Save attempt': {'Reflexes': (0, .15, 0, 0),
                     'Save attempt': (0, .1, 0, 0)
                     },
    'Shot': {
        'Shot': (.15, 0, 0, 0)
    }
}


def get_event_scores(etype: str, esubtype: str)
-> Tuple[float, float, float, float]:
    subtype_scores = activity_scores.get(etype)
    if subtype_scores is None:
        return 0, 0, 0, 0
    elif type(subtype_scores) is dict:
        return subtype_scores.get(esubtype, (0, 0, 0, 0))
    else:
        return subtype_scores


def cal_act_lvls(time_series_data: pd.DataFrame,
                 match_id: int or None,
                 player_id: str or None,
                 team_id: str = 'Huskies',
                 interval_seconds: float = 60.0)
                 -> np.ndarray or None:
    timer = 0
    curr_act_lvl = np.zeros(4, dtype=float)
    act_lvls = []
    pcond = True if player_id is None else
        time_series_data['OriginPlayerID'] == player_id
    mcond = True if match_id is None
        else time_series_data['MatchID'] == match_id
    data = time_series_data[pcond & mcond &
                            (time_series_data['TeamID'] == team_id)]
    if len(data) == 0:
        print("Warning: no data for player #{0} in match #{1}"
            .format(player_id, match_id))
        return None
    player_time = data['EventTime'].to_numpy()
    etypes = data['EventType']
    esubtypes = data['EventSubType']
    for i in range(data.shape[0] - 1):
        interval = player_time[i + 1] - player_time[i]
```

```
        timer += interval
        if timer > interval_seconds:
            data, reset timer
            act_lvls.append(curr_act_lvl.copy())
            timer = interval
            curr_act_lvl.fill(0)
        else:
            curr_act_lvl += np.asarray(get_event_scores(
                etypes.iloc[i], esubtypes.iloc[i]))
            timer += interval
    return np.asarray(act_lvls)


def get_activity_level():
    all_events.loc[all_events['MatchPeriod'] == '2H', 'EventTime']
    += 45 * 60
    all_events.sort_values('EventTime', inplace=True)

    types = ['attack', 'defense', 'collaborate', 'foul']
    df_dict = {'MatchID': match_ids}
    for i in range(4):
        df_dict['huskies_mean_' + types[i]] = []
        df_dict['huskies_std_' + types[i]] = []
        df_dict['oppo_mean_' + types[i]] = []
        df_dict['oppo_std_' + types[i]] = []

    for mi in match_ids:
        match_data = matches_df[matches_df['MatchID'] == mi]
        outcome = match_data['Outcome'].to_list()[0]
        oppo_team_id = match_data['OpponentID'].to_list()[0]

        shots = all_events[(all_events['EventType'] == 'Shot') &
            (all_events['MatchID'] == mi)]
        huskies_shots_time = shots[shots['TeamID'] == 'Huskies']
            ['EventTime']
        oppo_shots_time = shots[shots['TeamID'] == oppo_team_id]
            ['EventTime']

        huskies_act_lvls = cal_act_lvls(all_events, mi, player_id=None,
            team_id='Huskies')
        oppo_act_lvls = cal_act_lvls(all_events, mi, player_id=None,
            team_id=oppo_team_id)

        oppo_len = oppo_act_lvls.shape[0]
        huskies_len = huskies_act_lvls.shape[0]
        if huskies_len < oppo_len:
            huskies_act_lvls = np.pad(huskies_act_lvls, ((0, oppo_len
            - huskies_len), (0, 0)), 'constant')
        else:
            oppo_act_lvls = np.pad(oppo_act_lvls, ((0, huskies_len
            - oppo_len), (0, 0)), 'constant')

        for i in range(4):
            df_dict['huskies_mean_' + types[i]].append(np.mean(
                huskies_act_lvls[:, i]))
```

```
            df_dict['huskies_std_' + types[i]].append(np.std(
                huskies_act_lvls[:, i]))
            df_dict['oppo_mean_' + types[i]].append(np.mean(
                oppo_act_lvls[:, i]))
            df_dict['oppo_std_' + types[i]].append(np.std(
                oppo_act_lvls[:, i]))
    df = pd.DataFrame(df_dict)
    df.set_index('MatchID', inplace=True)
    return df
```

models.py

```
def kfold_model(x, y, model, params: dict, n_splits: int, random_state: int):
    from sklearn.model_selection import KFold
    kf = KFold(n_splits=n_splits)
    for train_index, test_index in kf.split(df):
        m = model(**params, random_state=random_state)
        X_train, X_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]
        m.fit(X_train, y_train)
        print(m.score(X_test, y_test))
```

network.py

```
from soccer_network.data import (match_ids, matches_df)
from soccer_network.graphs import load_graphml
from graph_tool import Graph, clustering,
    correlations, centrality, VertexPropertyMap
import numpy as np
from typing import Tuple, List, Callable, Any

class Network:
    def __init__(self):
        self.g = Graph(directed=True)
        self.player_id_to_vertex = {}
        self.pairs = {}
        self.g.vertex_properties['player_id'] =
            self.g.new_vertex_property("string")
        self.g.vertex_properties['player_coords'] =
            self.g.new_vertex_property("vector<float>")
        self.g.vertex_properties['average_player_coords'] =
            self.g.new_vertex_property("vector<float>")
        self.g.vertex_properties['player_n_coords'] =
            self.g.new_vertex_property("int")
        self.g.edge_properties['weight'] =
            self.g.new_edge_property("float")

    @property
    def edge_weights(self):
        return self.g.edge_properties['weight']
```

```
    @property
    def player_id_pmap(self):
        return self.g.vertex_properties['player_id']

    @property
    def player_coords_pmap(self):
        return self.g.vertex_properties['player_coords']

    @property
    def player_n_coords_pmap(self):
        return self.g.vertex_properties['player_n_coords']

    @property
    def average_player_coords_pmap(self):
        # lazy evaluation of means
        for v in self.g.vertices():
            self.g.vertex_properties['average_player_coords'][v] =
            np.asarray(
                self.player_coords_pmap[v]) /
                self.player_n_coords_pmap[v]
        return self.g.vertex_properties['average_player_coords']

    def add_players(self, pids: List[str]):
        n = len(pids)
        vs = list(self.g.add_vertex(n))
        self.player_id_to_vertex.update(
            {pids[i]: vs[i] for i in range(n)})
        for i in range(n):
            self.player_id_pmap[vs[i]] = pids[i]
        return vs

    def add_passes(self, id_pairs: List[Tuple],
        coords_pairs: List[Tuple], pass_scores=None):
        pairs = [(self.player_id_to_vertex[i1],
            self.player_id_to_vertex[i2])
                 for i1, i2 in id_pairs]
        n = len(coords_pairs)
        if pass_scores is None:
            pass_scores = [1 for _ in range(n)]

        for i in range(n):
            coords = self.player_coords_pmap[pairs[i][0]]
            if len(coords) == 0:
                coords = np.asarray([coords_pairs[i][0],
                    coords_pairs[i][1]])
            else:
                coords += np.asarray([coords_pairs[i][0],
                    coords_pairs[i][1]])
            self.player_coords_pmap[pairs[i][0]] = coords
            self.player_n_coords_pmap[pairs[i][0]] += 1

            coords = self.player_coords_pmap[pairs[i][1]]
            if len(coords) == 0:
                coords = np.asarray([coords_pairs[i][2],
                    coords_pairs[i][3]])
```

```
            else:
                coords += np.asarray([coords_pairs[i][2],
                    coords_pairs[i][3]])
            self.player_coords_pmap[pairs[i][1]] = coords
            self.player_n_coords_pmap[pairs[i][1]] += 1

            e = self.pairs.get(pairs[i])
            if e is not None:
                self.edge_weights[e] += pass_scores[i]
            else:
                e = self.g.add_edge(*pairs[i])
                self.pairs[pairs[i]] = e
                self.edge_weights[e] = pass_scores[i]

    def cleanup(self):
        """remove isolated vertices"""
        to_remove = []
        for v in self.g.vertices():
            if v.in_degree() + v.out_degree() == 0:
                to_remove.append(v)
        n = len(to_remove)
        self.g.remove_vertex(to_remove, fast=True)
        print("Removed {0} isolated vertices".format(n))

    def save(self, file: str):
        self.g.save(file, fmt='graphml')


def post_results_as_vertex_properties(results:
        List[VertexPropertyMap]):
    return [np.asarray(r.a) for r in results]


def clustering_coefficient(g: Graph):
    return clustering.local_clustering(g,
        weight=g.edge_properties['weight'], undirected=False)


def assortativity(g: Graph):
    return correlations.assortativity(g, 'total',
        g.edge_properties['weight'])


def post_assortativity(results):
    mean, variance = zip(*results)
    return mean + variance


def pagerank_centrality(g: Graph):
    return centrality.pagerank(g, weight=
        g.edge_properties['weight'])


def closeness_centrality(g: Graph):
    return centrality.closeness(g, weight=
```

```
        g.edge_properties['weight'])


def betweenness_centrality(g: Graph):
    return centrality.betweenness(g, weight=
        g.edge_properties['weight'])[0]


def eigenvector_centrality(g: Graph):
    return centrality.eigenvector(g, g.edge_properties['weight'])[1]


def katz_centrality(g: Graph):
    return centrality.katz(g, weight=g.edge_properties['weight'])


post_pagerank_centrality = \
    post_eigenvector_centrality = \
    post_beweenness_centrality = \
    post_closeness_centrality = \
    post_clustering_coefficient = \
    post_katz_centrality = \
    post_results_as_vertex_properties

metrics: List[Callable] = [
    passing_volume,
    clustering_coefficient,
    pagerank_centrality,
    closeness_centrality,
    betweenness_centrality,
    eigenvector_centrality,
    katz_centrality,
]

post_metrics: List[Callable] = [
    post_passing_volume,
    post_clustering_coefficient,
    post_pagerank_centrality,
    post_closeness_centrality,
    post_beweenness_centrality,
    post_eigenvector_centrality,
    post_katz_centrality,
]


def run_metric(gs: List[Graph], metric:
        Callable[[Graph], Any], post_metric: Callable) -> List[np.ndarray]:
    results = [metric(g) for g in gs]
    return post_metric(results)


if __name__ == "__main__":
    print('Loading graphml files...')
    graphs = [load_graphml('../graphs/network-{0}.xml'.format(mi))
        for mi in match_ids]
```

```
print('Calculating metrics...')
for m, pm in zip(metrics, post_metrics):
    run_metric(graphs, m, pm)
```

	Attack	Defense	Collaborate	Foul
Air duel	0.2	0	0	0
Ground attacking duel	0.15	0	0	0
Ground defending duel	0	0.15	0	0
Ground loose ball duel	0.1	0	0	0
Foul	0	0	0	0.1
Hand foul	0	0	0	0.3
Late card foul	0	0	0	0.1
Out of game foul	0	0	0	0.3
Protest	0	0	0	1.0
Simulation	0	0	0	0.1
Time lost foul	0	0	0	0.1
Violent Foul	0.05	0	0	0.15
Corner	0.1	0	0	0
Free Kick	0.1	0	0	0
Free kick cross	0.1	0	0.1	0
Free kick shot	0.15	0	0	0
Goal kick	0.2	0	0	0
Penalty	0.1	0	0	0
Throw in	0.2	0	0	0
Goalkeeper leaving line	0	0	0	0.2
Ball out of the field	0	0	0	0
Whistle	0	0	0	0
Acceleration	0	0	0	0
Clearance	0	0.15	0	0
Touch	0	0	0.15	0
Cross	0	0	0.15	0
Hand pass	0	0	0.1	0
Head pass	0	0	0.15	0
High pass	0	0	0.15	0
Launch	0	0	0.1	0
Simple pass	0	0	0.1	0
Smart pass	0	0	0.2	0
Reflexes	0	0.15	0	0
Save attempt	0	0.1	0	0
Shot	0.15	0	0	0
Offside	0	0	0	1

Table 6: Weights for different sub event type.