令和 6 年度　特別研究報告書

機械学習によるサッカーの戦術分析

龍谷大学　先端理工学部　知能情報メディア課程

Y210121 奥村星哉

指導教員　三好力　教授

内容梗概

データ解析技術の進展により、スポーツ分野でのデータ活用が注目を集める中、特にサッカーでは選手のトラッキングデータを活用した分析が進んでいる。これにより、選手の走行距離やポジショニング、パス成功率やシュート精度といった技術的・身体的パフォーマンスを評価することが可能。しかし、こうした技術の普及や試合状況に応じた判断支援手法の整備は十分ではない。リアルタイム処理や戦術への活用における技術的課題も多く、現場でのデータ活用の可能性が十分に引き出されていない現状がある。
本研究では、サッカー試合におけるトラッキングデータとイベントデータを基に、攻撃がシュートに至るかを予測するモデルを構築する。このモデルにより、シュート成功に関わるパターンや要素を特定し、重要なアクションを抽出することを目指す。選手のポジショニングやパス軌道などの動的データを活用し、得点機会を増加させる指針を提供します。また、モデルの精度向上により信頼性を高め、指導者や選手が具体的な戦術行動を選択するための効果的なデータ活用を実現する。これにより、サッカー界でのデータ活用の普及に寄与することを目的としている。

目次

1, はじめに

データ解析の進歩や計測技術の発展に伴い、様々なスポーツにおいてデータを活用した評価手法が大きな影響を与えている。特にチームスポーツにおいては、試合中の選手の動きなど様々なデータが収集され、チームのパフォーマンスを評価し、強化するためにデータを利用することが必要不可欠となっている。サッカーにおいては、選手のトラッキングデータを用いることで、選手の走行距離、スプリント数、ポジショニングなどを分析できる。このデータにより、選手のスタミナやフィットネスレベルを把握することが可能である。また、パス成功率、シュート精度、タックル成功率などの統計を収集し、選手の技術的なパフォーマンスを評価するための重要な指標として活用されている。

しかし、このようなデータ活用の技術がサッカー界全体で広く普及しているとは言えない状況にある。特に、トラッキングデータやイベントデータを解析し、試合の状況に応じた具体的な判断を支援するための手法がまだ十分に整備されていないことが課題として挙げられる。また、試合中にリアルタイムでデータを処理し、即座にフィードバックを得るためのシステムやモデルの構築には多くの技術的な課題が存在している。さらに、データ解析結果を現場で戦術的に活用するための具体的な指針が不足しているため、データ活用の可能性が十分に引き出されていない状況にある。

本研究では、このような課題を解決するため、サッカーの試合における選手のトラッキングデータとイベントデータを活用し、特定の攻撃がシュートに至るか否かを予測するモデルを構築することを目的とする。このモデルにより、シュートに至る攻撃に共通する特定の要素やパターンを明らかにし、攻撃の成功を左右する重要なアクションを抽出することが可能となる。具体的には、選手のポジショニングやパスの軌道、相手選手との距離などの動的データを基に、シュートにつながる確率の高い場面を見極め、得点機会を効果的に増加させるための指針を導くことを目指す。従来の統計的手法や戦術分析に比べ、トラッキングデータを活用した本モデルは、リアルタイムで選手の動きと連携を考慮できるため、より精度の高い戦術の最適化が可能である。

本研究の目的は、指導者や選手が攻撃の過程で注意すべきパス選択、スペースの使い方、フォーメーションの調整といった具体的な行動を特定することである。これにより、試合の状況に応じた効果的な戦術の指針を提供し、データの活用を普及させるための足掛かりとする。また、予測モデルの精度を向上させることで、その信頼性を高めることも本研究の重要な目的である。

2，既存技術

2.1　得点期待値（xG）モデル

xG（Expected Goals）[1]モデルは、各シュートがどの程度の確率で得点に結びつくかを予測する指標である。シュートの距離、角度、シュートを行った選手や守備陣形などの特徴を基に、得点確率を算出する。このモデルは、ロジスティック回帰やランダムフォレスト、XGBoost などの機械学習アルゴリズムを用いて構築されることが一般的であり、試合や選手のパフォーマンス評価において広く用いられている。xG モデルは、得点機会の質を測るための標準的な指標であり、戦術分析や選手評価において重要な役割を果たしている。

2.2 PV（Possession Value）モデル

Opta が提供する PV（Possession Value）[2]は、得点の可能性を評価する指標であり、ボールを保持しているチームが次の 10 秒間で得点する確率を算出するモデルである。PV モデルは、トラッキングデータやイベントデータを用いて構築され、ボール保持者や周囲の選手の位置関係、相手の守備配置、プレーの方向性やスピードといった要因を考慮している。PV は、シュートだけでなく、ボール保持やパス、ポジショニングが得点機会にどのように貢献しているかを総合的に評価するためのモデルであり、特に攻撃の成否を評価する際に役立つ。

2.3　ロジスティック回帰

ロジスティック回帰[3]は、サッカーにおける得点やシュートの成功、失敗を予測するために広く用いられている統計モデルである。「得点が生まれるか」「攻撃がシュートに至るか」といった 2 値分類に適しており、シュートの角度、距離、選手のポジション、ディフェンダーの位置などの特徴量を基に予測を行う。ロジスティック回帰の利点は、モデルの解釈が容易であり、各特徴量が予測結果に与える影響を係数として示せる点である。このため、ロジスティック回帰は、試合の状況に応じたパフォーマンスの傾向を把握するのに有用である。

2.4 RNN（Recurrent Neural Network）

RNN（Recurrent Neural Network）[4]は、時系列データやシーケンスデータを処理するために設計されたニューラルネットワークの一種である。通常のフィードフォワード型ニューラルネットワークとは異なり、過去の情報を保持して、それを現在の出力に影響させることができる。これにより、文脈や連続性のあるデータ（例えば、時間軸で変化するデータ）を学習するのに適している。

2.5 LSTM（Long Short-Term Memory）

LSTM（Long Short-Term Memory）[5]は、RNN の一種であり、時系列データや連続したデータの学習に特化したニューラルネットワークモデルである。特に、長期的な依存関係を捉えることができる点で、従来の RNN より優れている。LSTM は、セル状態（記憶）を保持し、入力ゲート、忘却ゲート、出力ゲートを用いることで、重要な情報を選別しながら情報を長期間保持できる仕組みを持っている。

3，提案手法

PV モデルは実用化されているものの、精度や信頼性が不足しているため、広く普及に至っていないことが課題である。本研究では、サッカーの試合における攻撃の成否を予測し、有効なアクションを抽出するために以下の手法を提案する。本手法では、データの抽出と加工、学習モデルの構築・評価を通じて、最も精度の高い予測モデルを選定し、それに基づいて攻撃成功に寄与するアクションを特定する。

具体的には、過去の試合から収集したトラッキングデータおよびイベントデータを使用する。このデータには、選手の位置情報、ボール保持者の座標、パス、ドリブル、シュートなどの配置が含まれる。試合中の攻撃シーンをトラッキングデータとイベントデータを基に識別し、一連のプレーシーケンスとして抽出する。各シーケンスを数値データとして扱えるよう、特徴量を計算・加工する。

シーケンスごとにシュートの発生を予測する分類モデルを構築する。使用するモデルとして、特徴量とシュート発生の関係を単純かつ解釈しやすい形でモデル化するロジスティック回帰と、シーケンスデータを扱い時系列情報を考慮した高精度な予測が可能なリカレントニューラルネットワーク（RNN）を採用する。抽出したデータを学習データとしてそれぞれのモデルに訓練を行い、交差検証法を用いてモデルの性能を評価する。シュート発生予測において最も精度の高いモデルを選定する。

選定した最良モデルを使用し、攻撃イベントごとにシュート確率を計算する。イベント直前と直後のシュート確率の変化を比較し、確率上昇が最大となるイベントを特定する。シュート確率を大きく向上させたアクション（パス、ドリブル、ポジショニングなど）を「有効なアクション」として抽出する。抽出したアクションの特徴（位置、タイミング、方向性など）を分析し、攻撃成功に共通する要素を特定する。

さらに、抽出した有効アクションを基に、攻撃成功に寄与するプレーの特徴を明らかにする。チームや選手に対して具体的な戦術的改善案や個別指導の指針を提供する。また、分析結果を過去の試合や実験データに適用し、提案手法の有効性を定量的に評価する。最後に、攻撃成功率やシュート頻度の向上が見られるかどうかを検証する。

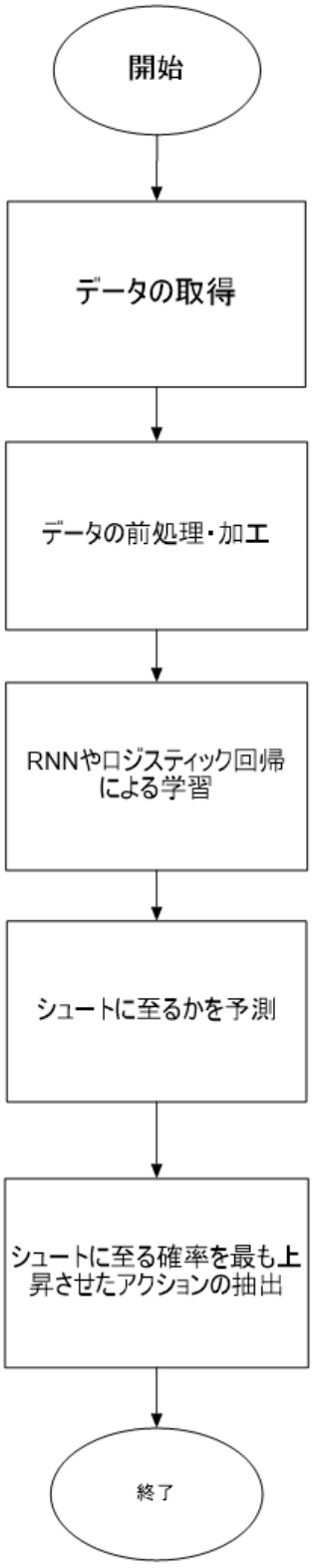

図 3.1　本システムのフローチャート

4, 実験

4.1　PV を参考にしたシュート確率予測と攻撃有効アクションの抽出

本手法では、トラッキングデータおよびイベントデータを基に、攻撃がシュートに至る可能性を評価する「シュート確率モデル」を構築する。これにより、攻撃がどの位置・タイミングでシュートにつながりやすいかを予測し、シュート確率を最大化するための有効なアクション を抽出する。

4.2 シュート確率予測モデルの構築

4.2.1 データの前処理と特徴量の設定

ロジスティック回帰に使用するデータはトラッキングデータ（図 1 に示す）とイベントデータ(図 2 に示す)でイベントデータによりシュート（SHOT）とボールを失うシーン（BALL　OUT、BALL　LOST）の抽出し、その瞬間の選手全員の座標とボールの座標、加えて 10 秒前の選手全員の座標とボールの座標を特徴量として抽出する。

RNN に使用するデータは攻撃ごとにトラッキングデータを配列に格納し、0.04 秒ごとの選手全員とボールの座標を特徴量とする。

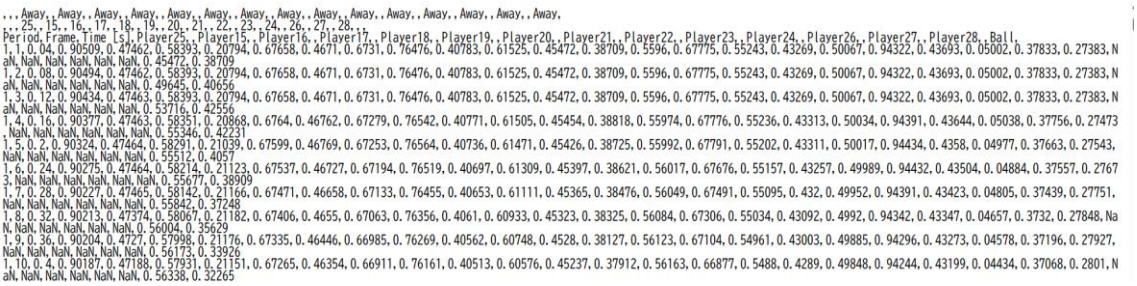

図 1 トラッキングデータ

1	Team	Type	Subtype	Period	Start Frame	Start Time [s]	End Frame	End Time [s]	From	To	Start X	St
2	Away	SET PIECE	KICK OFF	1	1	0.04	0	0	Player19		NaN	N
3	Away	PASS		1	1	0.04	3	0.12	Player19	Player21	0.45	0.
4	Away	PASS		1	3	0.12	17	0.68	Player21	Player15	0.55	0.
5	Away	PASS		1	45	1.8	61	2.44	Player15	Player19	0.55	0.
6	Away	PASS		1	77	3.08	96	3.84	Player19	Player21	0.45	0.
7	Away	PASS		1	191	7.64	217	8.68	Player21	Player22	0.4	0.
8	Away	PASS		1	279	11.16	303	12.12	Player22	Player17	0.39	0.
9	Away	BALL LOST	INTERCEPTION	1	346	13.84	380	15.2	Player17		0.51	0.
10	Home	RECOVERY	INTERCEPTION	1	378	15.12	378	15.12	Player2		0.27	0.
11	Home	BALL LOST	INTERCEPTION	1	378	15.12	452	18.08	Player2		0.27	0.

（b）イベントデータ

4.2.2 データの処理・加工

トラッキングデータに含まれる欠損値はゼロで補完した。また、時刻情報（Time [s]）を基準にデータを昇順に並び替えた。

Home チームと Away チームのトラッキングデータの時刻（Time [s]）をキーとして統合し、1 つのデータフレームにまとめた。この統合により、両チームの選手座標が同時に処理できる形式とした。

攻撃終了イベントの定義はイベントデータ中の「BALL LOST」「BALL OUT」「SHOT」の 3 種類を攻撃終了イベントと定義し、各イベントの開始時刻（Start Time [s]）を抽出した。
攻撃開始・終了時刻のペア作成は試合開始時刻（0 秒）を最初の攻撃開始時刻とし、各攻撃終了時刻をもとに攻撃開始・終了のペアを作成した。
攻撃区間のトラッキングデータの抽出は攻撃開始時刻から終了時刻までのトラッキングデータを抽出し、攻撃ごとに区間分割を行った。

攻撃データの形状は攻撃区間ごとに、トラッキングデータを以下の形状で整備した：
(攻撃数,タイムステップ数,特徴量数)
本研究では、攻撃数 332、タイムステップ 4344、特徴量数 64（Home チームと Away チームの選手およびボール座標）となった。
ラベル付けは各攻撃区間に対し、攻撃が「SHOT」で終了した場合を 1（成功)、それ以外を 0（失敗）として二値ラベルを付与した。

4.2.3 モデルの学習

シュート確率の予測にはロジスティック回帰と RNN 用い、過去のシュート成功・失敗のデータを用いてモデルを学習させる。

使用したモデル

本研究では、トラッキングデータの時系列性を捉えるために、LSTM（Long Short-Term Memory）ネットワークを採用した。以下のモデル構造を設計した：

入力層：最大タイムステップ数 4344、特徴量数 64 のデータを入力。
隠れ層：LSTM 層を 2 層（それぞれ 64 ユニット、32 ユニット）重ね、時系列の重要な特徴を抽出。
出力層：1 ユニットの Dense 層（シグモイド活性化関数）で成功（1）または失敗（0）の二値分類。

損失関数と評価指標

損失関数：Binary Cross-Entropy を使用。
評価指標：Accuracy に加え、クラス不均衡を考慮し F1 スコアを採用。

クラス重み

ゴール成功（SHOT）のデータが少ないため、クラス 1（成功）の重みを高く設定してモデルのバランスを調整。

4.2.4 モデルの評価と改善:

モデルごとにトレーニングデータ（80%）とテストデータ（20%）を用いて学習および評価。
評価指標として、正解率（Accuracy)、適合率（Precision)、再現率（Recall)、F1 スコアを用いて精度を比較。

4.3 攻撃に有効なアクションの抽出

4.3.1　シュート確率が高まる状況の分析:

モデルにより算出されたシュート確率を用いて、特定の位置・タイミング・パス選択がシュート発生に有利かどうかを分析する。これにより、どのエリアやプレーがシュートにつながりやすいかを特定する。

4.3.2 有効なアクションの特定:

シュート確率が上がるプレーやアクションを統計的に解析し、共通する要因を抽出する。たとえば、「ゴール前での短いパス交換がシュートにつながりやすい」「相手ディフェンダーを引き寄せてからの横パスが有効」などの特徴的なアクションが見つかる場合がある。

モデルの予測を基に、予測値が最も増加したアクションをイベントデータから抽出し、アニメーションとして出力する。

5, 結果

まずは、ロジスティック回帰と RNN によるアンダーサンプリングの有無で分けて分析を行った結果を示す。

表 5.1　ロジスティク回帰による分析（アンダーサンプリングあり）

Metric	Class 0	Class 1	Macro Avg	Weighted Avg
Precision	0.93	0.20	0.56	0.86
Recall	0.87	0.33	0.60	0.82
F1-Score	0.90	0.25	0.57	0.84
Support	61	6	67	67

表 5.2　ロジスティック回帰による分析（アンダーサンプリングなし）

Metric	Class 0	Class 1	Macro Avg	Weighted Avg
Precision	0.94	0.00	0.47	0.88
Recall	1.00	0.00	0.50	0.94
F1-Score	0.97	0.00	0.48	0.91
Support	79	5	84	84

表 5.3　RNN による分析（アンダーサンプリングあり）

Metric	Class 0	Class 1	Macro Avg	Weighted Avg
Precision	0.97	0.31	0.64	0.85
Recall	0.55	0.92	0.73	0.61
F1-Score	0.70	0.46	0.58	0.65
Support	55	12	67	67

表 5.4　RNN による分析（アンダーサンプリングなし）

Metric	Class 0	Class 1	Macro Avg	Weighted Avg
Precision	0.81	0.17	0.49	0.69
Recall	0.45	0.50	0.48	0.46
F1-Score	0.58	0.25	0.42	0.52
Support	55	12	67	67

5.1. ロジスティック回帰（アンダーサンプリングあり）

- **Class 0（Not Shot）**:

  Precision、Recall、F1-Score が全て高い（Precision: 0.93, Recall: 0.87, F1: 0.90）ため、クラス 0 の予測性能は良好である。

- **Class 1（Shot）**:

  Precision（0.20）や F1-Score（0.25）は低いが、Recall（0.33）が比較的高い。

  少数クラスである Class 1 をある程度認識しているものの、Precision の低さから False Positives が多いことが示唆される。

- **全体的なパフォーマンス**:

  Macro Avg や Weighted Avg のスコアはバランスが取れている（F1: Macro 0.57, Weighted 0.84）。アンダーサンプリングにより少数クラスのバランスが改善されましたが、少数クラスの Precision 改善が課題である。

5.2. ロジスティック回帰（アンダーサンプリングなし）

- **Class 0（Not Shot）**:

  Precision（0.94）と Recall（1.00）が非常に高いが Class 1 に偏ってしまっている。

- **Class 1（Shot）**:

  Precision、Recall、F1-Score 全てが 0.00。少数クラスをほとんど識別できていない。

- **全体的なパフォーマンス**:

  全体の Accuracy（0.94）が高く見えますが、これはクラス不均衡の影響によるものでクラス不均衡を補正しなければ、少数クラスの予測性能は向上しないと予想される。

5.3. RNN（アンダーサンプリングあり）

- **Class 0（Not Shot）**:

  Precision（0.97）は高いが、Recall（0.55）はやや低い。Class 0 のサンプルの多くを見逃している可能性。

- **Class 1（Shot）**:

  Recall（0.92）が非常に高く、F1-Score（0.46）もロジスティック回帰（アンダーサンプリングあり）と比較して改善。

  Precision（0.31）がやや低いが、少数クラスを予測する能力が向上している。

- **全体的なパフォーマンス**:

  Macro Avg や Weighted Avg のスコアが高く、全体のバランスが改善。特に Recall が全体で 0.61（Weighted Avg）と向上しており、少数クラスの認識に強み。

5.4. RNN（アンダーサンプリングなし）

- **Class 0（Not Shot）**:

  Precision（0.81）と Recall（0.45）が低下。クラス不均衡の影響でパフォーマンスが低下。

- **Class 1（Shot）**:

  Precision（0.17）や F1-Score（0.25）が非常に低い。Recall（0.50）は一定水準を保っているが、

Precision の低さから少数クラスの予測性能は課題である。

- **全体的なパフォーマンス**:

  Macro Avg と Weighted Avg のスコアが全て低下。クラス不均衡が RNN にも影響を与えている。

5.5 最も効果的なアクション抽出の結果

最も有効的だったアクションが何秒のどのアクションでシュートに至る確率と前のアクションからのシュート確率の変化の値を以下の表 5.5 と 5.6 に示す。さらに、抽出されたアクションをアニメーションにして表示したものを以下に図 5.1~5.6 に示す。

表 5.5 有効だったアクション

type	PASS
start_time	856.52
end_time	857.48
shot_probability	0.501772
change	0.026091

表 5.6 アクションごとのシュート確率の変化

type	start_time	end_time	shot_probability	change
RECOVERY	826.64	826.64	0.499769	NaN
PASS	826.64	828.64	0.438417	-0.061352
PASS	826.64	834.00	0.449023	0.010606
PASS	836.16	838.40	0.447760	-0.001262
PASS	842.52	843.44	0.456238	0.008477
PASS	845.12	846.48	0.484838	0.028600
PASS	851.76	852.72	0.457791	-0.027047
PASS	854.48	855.44	0.475681	0.017890
PASS	856.52	857.48	0.501772	0.026091
BALL LOST	857.48	858.92	0.493945	-0.007827

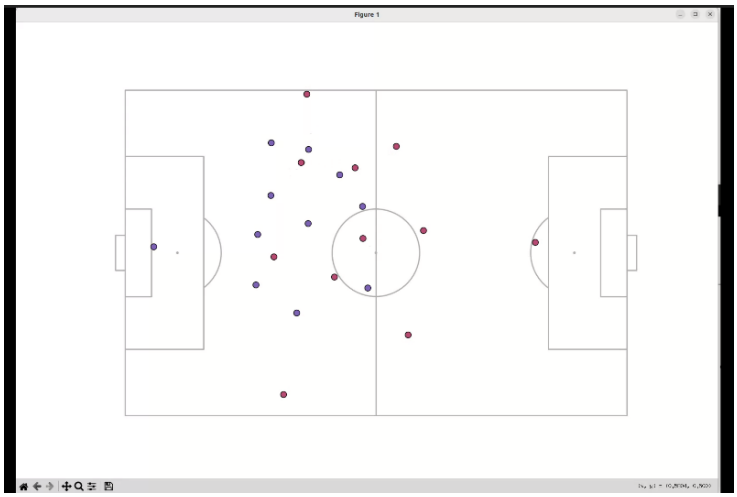

図 5. 1

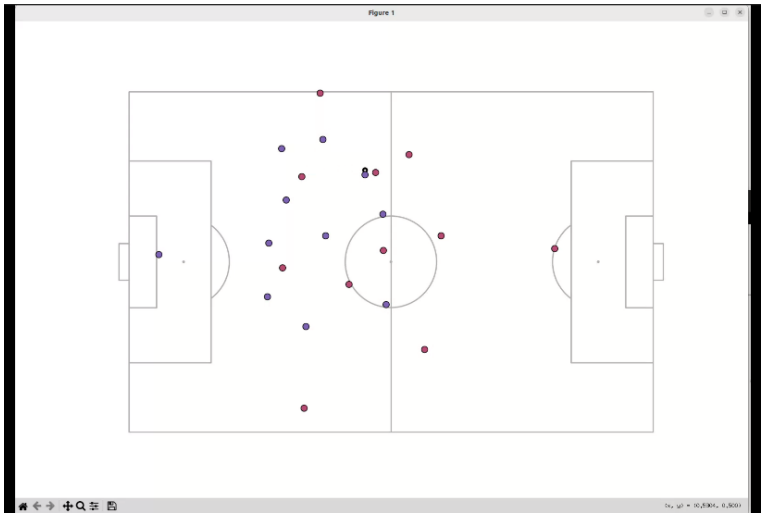

図 5.2

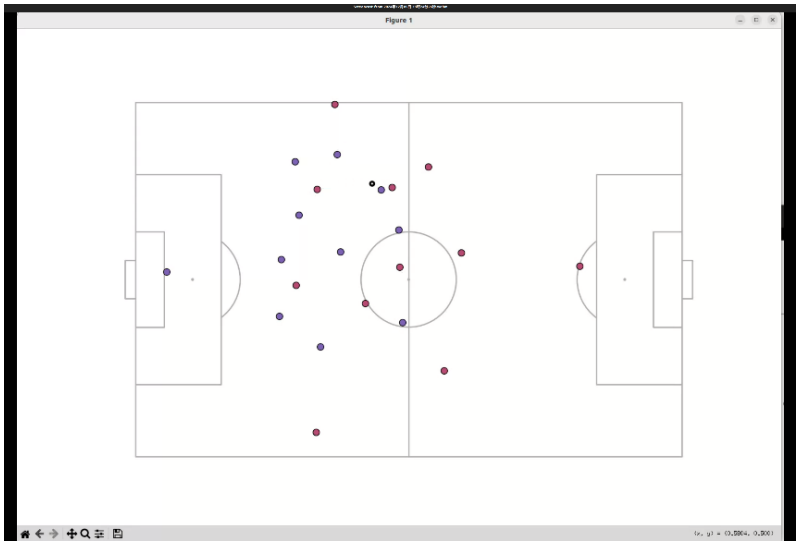

図 5.3

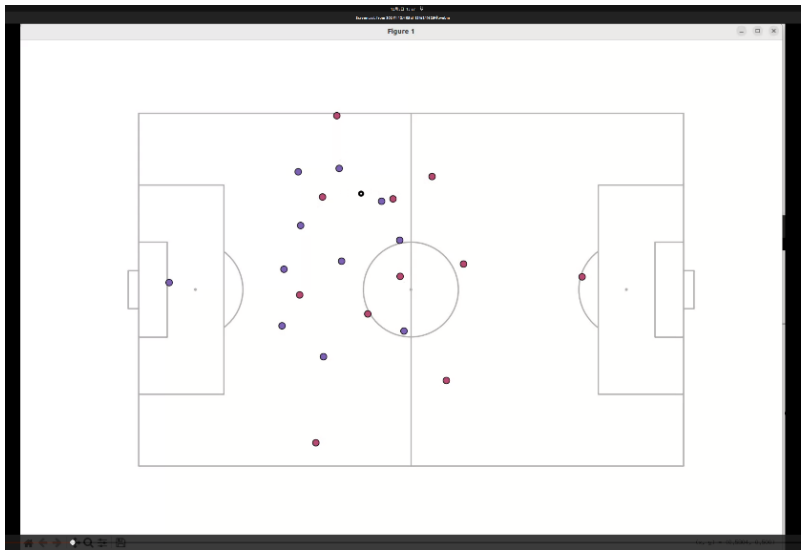

図 5.4

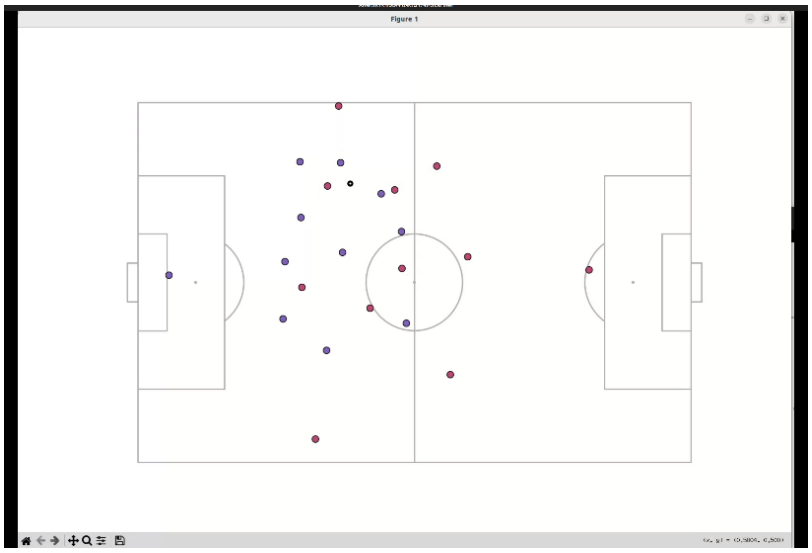

図 5.5

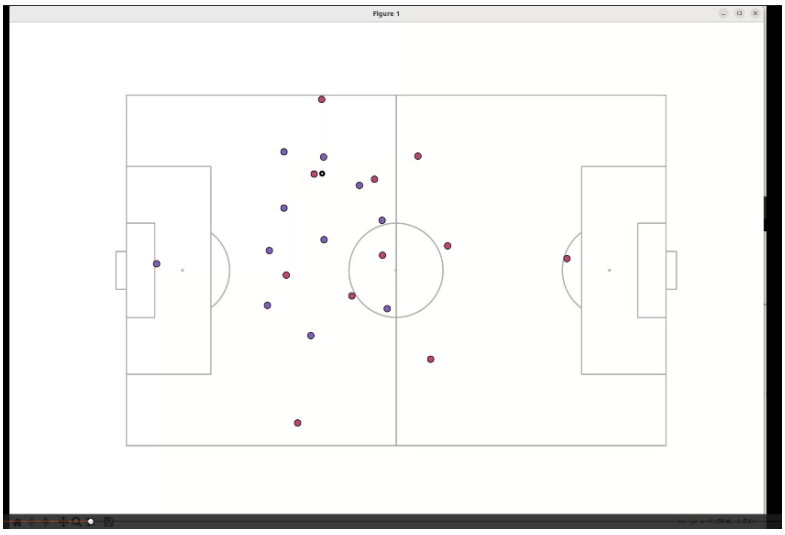

図 5.6

表 5.5 を見ると、有効アクションの開始時間が 856.52 秒から終了時間が 857.48 秒であり、シュートに至る確率が 0.501772 で前のアクションから 0.026091 上昇したことがわかる。
表 5.6 を見ると、各アクションの種類、開始時間と終了時間、シュートに至る確率と前のアクションからの変化の値が表されている。そのため、一つ目のパスは開始時間が 826.64 秒で終了時間も 828.64 秒でシュートに至る確率が 0.438417 で、前のアクションからの変化の値が-0.061352 なのでシュートに至る確率が 0.061352 下がったと言うことがわかる。
図 5.1~5.6 では、アクションを抽出しているのでセンターライン付近にいるボール保持者から縦パスを行っているシーンが抽出されたことがわかる。

6.考察

6.1 アンダーサンプリングの有効性:

表 5.1 と表 5.2、表 5.3 と表 5.4 をそれぞれ見比べるとアンダーサンプリングありの分析では、少数クラス（Class 1）の性能が大幅に改善している。Precision の低下が課題ですが、Recall が向上しており、少数クラスを捉える能力が改善されている。

6.2 RNN とロジスティック回帰:

表 5.1 と表 5.3 から RNN は特に Recall において優位性があり、少数クラスをより正確に検出している。表 5.2 と表 5.4 からアンダーサンプリングなしの RNN のパフォーマンスはクラス不均衡の影響で低下しており、サンプリング手法を活用する必要がある。

6.3　改善案

表 5.1 と 5.3 の Precision を改善するためにクラス 1（Shot）のサンプル数を増やしデータの増強をする。または recision-Recall カーブを利用して、最適な閾値を探索閾値の最適化を試みる。そして、RNN のさらなるパラメータ調整やモデルの改良を試みる。これにより、少数クラスの予測性能向上とモデル全体のバランス改善が期待できる。

6.4　有効的なアクションの抽出に対する考察

抽出されたアクションはサッカーの戦術的において妥当性が高いと評価できる。特に、抽出されるアクションは縦パスが多かったため、縦パスに関するアクションは、守備を崩す上で効果的であることが示唆された。

また、RNN を用いた手法が時系列情報を考慮した精度の高いモデルであり、アクション抽出に有効であった。

一方で、守備側の動きを考慮したさらなるモデル改良が必要であることや、使用データの偏りが結果に影響を与えた可能性も指摘される。今後は、異なるリーグやリアルタイム分析への応用を通じて、本手法の一般化と実践的有用性を検証していくことが必要である。

7.まとめ

本研究では、サッカーにおける攻撃の成否を予測し、有効なアクションを抽出する手法を提案した。トラッキングデータおよびイベントデータを活用し、シュート確率の変化を基に最も効果的なアクションを特定するため、ロジスティック回帰と RNN（リカレントニューラルネットワーク）を用いた予測モデルを構築した。実験を通じて、提案手法が具体的なアクションを抽出し、サッカー戦術における重要な示唆を提供できることを示した。

その結果、有効なアクションとして、縦パスやポジショニングの調整がシュート成功確率の向上に寄与することが明らかになった。

さらに、抽出されたアクションをアニメーション化し、具体的なプレーの流れを可視化することで、選手や指導者に対する実践的な指針を提供できる可能性を示した。

本研究の成果は、データ解析を基にした戦術の最適化や選手のトレーニング支援に寄与するものである。特に、データ解析の結果を視覚化し、指導現場で活用することで、データ駆動型の戦術設計が現実の試合において有効であることを示した。

一方で、本研究にはいくつかの限界が存在する。例えば、データの偏り（特定のリーグやチームに依存したデータ）や、守備側の要素を十分に考慮できていない点が挙げられる。今後は、異なるリーグや試合データを用いた検証、さらには守備データを含めた分析を行うことで、手法の汎用性と実用性をさらに高めることが求められる。

以上より、本研究はサッカー戦術におけるデータ解析の可能性を示すとともに、攻撃の成功要因を定量的に評価するための新たな手法を提案した。この成果は、サッカーにおけるデータサイエンスの発展に寄与するとともに、今後の研究や実践への応用可能性を示唆している。

謝辞

本研究を進めるにあたり、多大なご指導とご助言を賜りました三好力教授に心より感謝申し上げます。また、日頃より研究活動を共にし、ディスカッションや実験環境の整備などで多くの支援をしてくださった研究室の皆様に心から感謝いたします。
ここに、本研究に関わった全ての皆様に感謝の意を表し、謝辞とさせていただきます。

参考文献

[1] サッカー xg｜xG データ徹底解説！サッカーの新指標で勝利への道 ｜Football Station｜フットボール・ステーション https://football-station.net/xg/
[2] サッカーにおける選手の得点への貢献度を可視化した指標「ポゼッションバリュー（PV)」とは何か ｜Stats Perform https://note.com/stats_perform_jp/n/ne141cd977d78?magazine_key=m678837fc4a6e
[3]中井悦治（2015）「IT エンジニアのための機械学習理論入門」株式会社技術評論社
[4] リカレントニューラルネットワーク（RNN）とは？利用用途と解決すべき課題 - 株式会社 ProFab https://profab.co.jp/what-is-recurrent-neural-network/
[5]我妻幸長（2018）「はじめてのディープラーニング」SB クリエイティブ株式会社

付録

```
import pandas as pd
from tensorflow.keras.utils import pad_sequences
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Masking,
Input
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,
classification_report
import matplotlib.pyplot as plt

# データロードと処理
LINK1 = ('https://raw.githubusercontent.com/metrica-
sports/sample-data/master/'
        'data/Sample_Game_1/Sample_Game_1_RawTrackingDat
a_Home_Team.csv')
df_home_game1 = pd.read_csv(LINK1, skiprows=2)
df_home_game1.fillna(0, inplace=True)
df_home_game1.sort_values('Time [s]', inplace=True)

LINK2 = ('https://raw.githubusercontent.com/metrica-
sports/sample-data/master/'
        'data/Sample_Game_1/Sample_Game_1_RawTrackingDat
a_Away_Team.csv')
df_away_game1 = pd.read_csv(LINK2, skiprows=2)
df_away_game1.fillna(0, inplace=True)
df_away_game1.sort_values('Time [s]', inplace=True)

LINK3 = ('https://raw.githubusercontent.com/metrica-
sports/sample-data/master/'
        'data/Sample_Game_1/Sample_Game_1_RawEventsData.c
sv')
df_event_game1 = pd.read_csv(LINK3)
df_event_game1.sort_values('Start Time [s]', inplace=True)

attack_end_event = ['BALL LOST', 'BALL OUT', 'SHOT']
attack_end_times =
df_event_game1[df_event_game1['Type'].isin(attack_end_event)
]['Start Time [s]'].values

attack_start_times = [0]  # 初期攻撃の開始
attack_start_times += list(attack_end_times[:-1])  # 終了時点か
ら次の開始を推測

attack_pairs = [(start, end) for start, end in
zip(attack_start_times, attack_end_times)]

tracking_segments = [
    df_home_game1[(df_home_game1['Time [s]'] >= start) &
(df_home_game1['Time [s]'] <= end)]
    for start, end in attack_pairs
]

attack_data = [segment.iloc[:, 3:].values for segment in
tracking_segments]
max_timesteps = max(len(data) for data in attack_data)
padded_attacks = pad_sequences(attack_data,
maxlen=max_timesteps, padding='post', dtype='float32')

event_filtered =
df_event_game1[df_event_game1['Type'].isin(attack_end_event)
].copy()

labels = np.array([
    1 if 'SHOT' in event_filtered[(event_filtered['Start Time [s]']
>= start) &
                                  (event_filtered['Start Time [s]'] <=
end)]['Type'].values else 0
    for start, end in attack_pairs
])

print(f"攻撃データの形状: {padded_attacks.shape}")
print(f"ラベルの形状: {labels.shape}")
from imblearn.under_sampling import RandomUnderSampler

# データ分割
X_train, X_test, y_train, y_test = train_test_split(padded_attacks,
labels, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=42)

# アンダーサンプリングの適用
# X_train を 2 次元に変換（必要に応じてリシェイプ）、再サンプ
リング後に元の形に戻します
X_train_flat = X_train.reshape(X_train.shape[0], -1)  # 平坦化
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled =
rus.fit_resample(X_train_flat, y_train)

# 元の形にリシェイプ
X_train_resampled = X_train_resampled.reshape(-1,
max_timesteps, X_train.shape[2])

print(f"再サンプリング後の訓練データの形状:
{X_train_resampled.shape}")
print(f"再サンプリング後のラベル分布:
{np.bincount(y_train_resampled)}")

# モデル構築
model = Sequential([
    Input(shape=(max_timesteps, padded_attacks.shape[2])),
    Masking(mask_value=0.0),
    LSTM(64, return_sequences=False),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
print(model.summary())

# 再サンプリング後のデータで訓練
history = model.fit(
    X_train_resampled, y_train_resampled,
    epochs=20, batch_size=32,
    validation_data=(X_val, y_val),  # 検証データはそのまま
    class_weight=None  # クラス重みを削除（データバランスが
調整済みのため）
)

# 学習結果の可視化
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
plt.title("Accuracy")
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Loss")
plt.legend()
plt.show()

# テストデータで評価
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"テストデータでの損失: {test_loss:.4f}")
print(f"テストデータでの精度: {test_accuracy:.4f}")

# 混同行列と詳細レポート
predicted_probabilities = model.predict(X_test)
predictions = (predicted_probabilities > 0.5).astype(int)

cm = confusion_matrix(y_test, predictions)
print("混同行列:¥n", cm)

print(classification_report(y_test, predictions))

def calculate_shot_probability(model, tracking_data, event_data,
start_time, end_time):
    """
    シュート確率を計算する関数
    - model: 学習済みのモデル
    - tracking_data: トラッキングデータ
    - event_data: イベントデータ
    - start_time: 攻撃の開始時間
    - end_time: 攻撃の終了時間
    """
    # 該当時間範囲のイベントを抽出
    actions = event_data[(event_data['Start Time [s]'] >=
start_time) &
                         (event_data['End Time [s]'] <= end_time)]

    results = []

    for _, action in actions.iterrows():
        # 該当時間のトラッキングデータを取得
        time_data = tracking_data[(tracking_data['Time [s]'] >=
```

```
action['Start Time [s]']) &
                        (tracking_data['Time [s]'] <= action['End
Time [s]'])]
        if len(time_data) == 0:
            continue

        # 特徴量を抽出
        features = time_data.iloc[:, 3:].values  # 選手やボールの座
標データ
        features = features[np.newaxis, :, :]  # バッチ次元を追加

        # モデルでシュート確率を予測
        shot_prob = model.predict(features)[0][0]  # 出力の 1 つ
目が確率

        # 結果を保存
        results.append({
            'type': action['Type'],
            'team': action['Team'],
            'start_time': action['Start Time [s]'],
            'end_time': action['End Time [s]'],
            'x': action['Start X'],
            'y': action['Start Y'],
            'end_x': action['End X'],
            'end_y': action['End Y'],
            'shot_probability': shot_prob
        })

    return pd.DataFrame(results)

# 攻撃の時間範囲
start_time = 826.64
end_time = 858.92

# シュート確率を計算
shot_probabilities = calculate_shot_probability(model,
df_home_game1, df_event_game1, start_time, end_time)

# 最も有効なアクションを特定
if not shot_probabilities.empty:
    most_effective_action =
shot_probabilities.loc[shot_probabilities['shot_probability'].idxm
ax()]
    print("最も有効だったアクション:")
    print(most_effective_action)
else:
    print("該当するアクションが見つかりませんでした。")

def visualize_most_effective_action(action):
    plt.figure(figsize=(10, 6))
    plt.title("Most Effective Action")

    if action['type'] == 'PASS':
        plt.arrow(action['x'], action['y'],
                  action['end_x'] - action['x'], action['end_y'] -
action['y'],
                  color='blue', head_width=1, alpha=0.7, label='Pass')
    elif action['type'] == 'SHOT':
        plt.scatter(action['x'], action['y'], color='red', s=100,
label='Shot')
    elif action['type'] == 'DRIBBLE':
        plt.scatter(action['x'], action['y'], color='green', s=100,
label='Dribble')

    plt.xlabel('X Coordinate')
    plt.ylabel('Y Coordinate')
    plt.legend()
    plt.grid()
    plt.show()

# 可視化
if 'most_effective_action' in locals():
    visualize_most_effective_action(most_effective_action)


from matplotlib import animation
from matplotlib import pyplot as plt
from mplsoccer import Pitch
from IPython import display

# データロード
LINK1 = ('https://raw.githubusercontent.com/metrica-
sports/sample-data/master/'
        'data/Sample_Game_1/Sample_Game_1_RawTrackingDat
a_Home_Team.csv')
df_home = pd.read_csv(LINK1, skiprows=2)
df_home.sort_values('Time [s]', inplace=True)

LINK2 = ('https://raw.githubusercontent.com/metrica-
sports/sample-data/master/'
        'data/Sample_Game_1/Sample_Game_1_RawTrackingDat
a_Away_Team.csv')
df_away = pd.read_csv(LINK2, skiprows=2)
df_away.sort_values('Time [s]', inplace=True)

LINK3 = ('https://raw.githubusercontent.com/metrica-
sports/sample-data/master/'
        'data/Sample_Game_1/Sample_Game_1_RawEventsData.c
sv')
df_event = pd.read_csv(LINK3)
df_event.sort_values('Start Time [s]', inplace=True)

# カラムを整形
def set_col_names(df):
    cols = list(np.repeat(df.columns[3::2], 2))
    cols = [col+'_x' if i % 2 == 0 else col+'_y' for i, col in
enumerate(cols)]
    cols = np.concatenate([df.columns[:3], cols])
    df.columns = cols

set_col_names(df_home)
set_col_names(df_away)

# アクションの抽出（最も有効なアクション）
start_time = 826.64
end_time = 858.92

actions = df_event[(df_event['Start Time [s]'] >= start_time) &
                   (df_event['End Time [s]'] <= end_time)]

# 最も有効なアクションの選択
if most_effective_action is None and not actions.empty:
    if not actions[actions['Type'] == 'PASS'].empty:
        most_effective_action = actions[actions['Type'] ==
'PASS'].iloc[-1]
        print("代わりに最も有効なパスを選択しました:",
most_effective_action)
    elif not actions[actions['Type'] == 'DRIBBLE'].empty:
        most_effective_action = actions[actions['Type'] ==
'DRIBBLE'].iloc[-1]
        print("代わりに最も有効なドリブルを選択しました:",
most_effective_action)
    else:
        print("有効なアクションが見つかりませんでした。")
        most_effective_action = None

# 有効なアクションが見つからなかった場合の対策
if most_effective_action is None:
    raise ValueError("該当する有効なアクションがありませんで
した。アニメーションを生成できません。")

# 時間範囲でフィルタリング
df_home = df_home[(df_home['Time [s]'] >=
most_effective_action['start_time']) &
                  (df_home['Time [s]'] <=
most_effective_action['end_time'])].copy()
df_away = df_away[(df_away['Time [s]'] >=
most_effective_action['start_time']) &
                  (df_away['Time [s]'] <=
most_effective_action['end_time'])].copy()

# ボールデータの処理
df_ball = df_home[['Period', 'Frame', 'Time [s]', 'Ball_x',
'Ball_y']].copy()
df_ball.rename({'Ball_x': 'x', 'Ball_y': 'y'}, axis=1, inplace=True)
df_home.drop(['Ball_x', 'Ball_y'], axis=1, inplace=True)
df_away.drop(['Ball_x', 'Ball_y'], axis=1, inplace=True)

def to_long_form(df):
    """
    横持ちデータを縦持ちに変換する関数
    """
    df = pd.melt(df, id_vars=df.columns[:3],
value_vars=df.columns[3:], var_name='player')
```

```
    df.loc[df.player.str.contains('_x'), 'coordinate'] = 'x'
    df.loc[df.player.str.contains('_y'), 'coordinate'] = 'y'
    df = df.dropna(axis=0, how='any')
    df['player'] = df.player.str[6:-2]
    df = (df.set_index(['Period', 'Frame', 'Time [s]', 'player',
'coordinate'])['value']
          .unstack()
          .reset_index()
          .rename_axis(None, axis=1))
    return df

# 縦持ちデータ整形
df_home = to_long_form(df_home)
df_away = to_long_form(df_away)

# 描画領域定義
fig, ax = plt.subplots(figsize=(16, 10.4))
pitch = Pitch(pitch_type='metricasports', pitch_width=68,
pitch_length=105, goal_type='box', goal_alpha=1)
pitch.draw(ax)

ball, = ax.plot([], [], 'o', ms=6, mfc='w', mec='black', mew=3)
home, = ax.plot([], [], 'o', ms=10, mfc='#7f63b8', mec='black')
away, = ax.plot([], [], 'o', ms=10, mfc='#b94b75', mec='black')

# グラフ更新関数
def update(i):
    frame = df_ball.iloc[i, 1]
    ball.set_data(df_ball.loc[df_ball.Frame == frame, 'x'],
                  df_ball.loc[df_ball.Frame == frame, 'y'])
    away.set_data(df_away.loc[df_away.Frame == frame, 'x'],
                  df_away.loc[df_away.Frame == frame, 'y'])
    home.set_data(df_home.loc[df_home.Frame == frame, 'x'],
                  df_home.loc[df_home.Frame == frame, 'y'])
    return ball, away, home

# アニメーション定義
anim = animation.FuncAnimation(fig, update,
frames=len(df_ball), interval=50, blit=True)

# アニメーションを表示
plt.show()

# アニメーションを保存
w = animation.PillowWriter(fps=20)
anim.save('most_effective_action.gif', writer=w)
```