

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Система аналітики футбольних матчів»

Виконав студент групи ІІЗм-24-1
БУБЕНКО Олексій Вікторович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БУБЕНКО Олексій Вікторович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Система аналітики футбольних матчів»**,

затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р. № 491/с**

Вихідні дані для роботи: **Об'єктом дослідження у магістерській роботі є футбольні матчі та турнірні чемпіонати як явище спортивної діяльності та їхні статистичні характеристики, що відображають результати поєдинків і підсумкові розташування команд у турнірних таблицях. Предметом дослідження у магістерській роботі є методи та алгоритми прогнозування результатів футбольних матчів і формування ймовірного прогнозу фінальної турнірної таблиці.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р. С.	22.09.2025р.	22.09.2025р.
2	Савіцький Р. С.	22.09.2025р.	22.09.2025р.
3	Савіцький Р. С.	22.09.2025р.	22.09.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота магістра складається з опису дослідження методів прогнозування результатів футбольних матчів та фінальної турнірної таблиці, розробки та порівняння моделей машинного навчання, створення системи ймовірнісного прогнозування на основі статистичних даних та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи складається з 88 сторінок, 29 ілюстрацій, 30 літературних джерел та 3 додатків.

Мета роботи полягає у створенні системи, яка прогнозує результат окремого футбольного матчу та ймовірність точного рахунку, а також визначає ймовірність підсумкового місця команди у фінальній турнірній таблиці на основі симуляції сезону. Для досягнення цієї мети було використано методи машинного навчання, зокрема ансамблеві алгоритми, нейронні мережі та ймовірнісне моделювання на основі розподілу Пуассона.

Отримані результати включають розробку та порівняння трьох моделей прогнозування окремих матчів, серед яких модель випадкового лісу продемонструвала найкращу якість за метриками Brier Score та Ranked Probability Score. Для прогнозування фінальної турнірної таблиці розроблено модель на основі рекурентної нейронної мережі LSTM, яка формує ймовірнісний розподіл підсумкових позицій команд.

Структура роботи включає аналіз існуючих методів прогнозування футбольних матчів, розробку та порівняння моделей машинного навчання, а також проектування та реалізацію вебзастосунку для практичного використання системи.

КЛЮЧОВІ СЛОВА: МАШИННЕ НАВЧАННЯ, ПРОГНОЗУВАННЯ ФУТБОЛЬНИХ МАТЧІВ, ТУРНІРНА ТАБЛИЦЯ, ВИПАДКОВИЙ ЛІС, LSTM, НЕЙРОННІ МЕРЕЖІ, РОЗПОДІЛ ПУАССОНА, BRIER SCORE, RANKED PROBABILITY SCORE.

ABSTRACT

The master's qualification work consists of a description of methods for predicting football match outcomes and final league standings, development and comparison of machine learning models, creation of a probabilistic forecasting system based on statistical data, and an explanatory note. The explanatory note to the qualification work consists of 88 pages, 29 illustrations, 30 literature sources and 3 appendices.

The aim of the thesis is to develop a system that predicts the outcome of individual football matches and the probability of exact scores, as well as determines the probability of a team's final position in the league table based on season simulation. To achieve this goal, machine learning methods were employed, including ensemble algorithms, neural networks, and probabilistic modeling based on the Poisson distribution.

The obtained results include the development and comparison of three models for predicting individual matches, among which the Random Forest model demonstrated the best quality according to Brier Score and Ranked Probability Score metrics. For predicting the final league table, a model based on the LSTM recurrent neural network was developed, which generates a probabilistic distribution of teams' final positions.

The structure of the thesis includes an analysis of existing methods for football match prediction, development, and comparison of machine learning models, as well as the design and implementation of a web application for practical use of the system.

KEYWORDS: MACHINE LEARNING, FOOTBALL MATCH PREDICTION, LEAGUE TABLE, RANDOM FOREST, LSTM, NEURAL NETWORKS, POISSON DISTRIBUTION, BRIER SCORE, RANKED PROBABILITY SCORE.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ	9
1.1 Постановка задачі.....	9
1.2 Огляд наявних аналогів	10
1.3 Вибір програмних засобів для реалізації.....	13
1.4 Огляд наявних методів, моделей, алгоритмів прогнозування.....	15
Висновки до розділу 1	20
РОЗДІЛ 2. РОЗРОБКА МОДЕЛЕЙ ПРОГНОЗУВАННЯ РЕЗУЛЬТАТІВ ФУТБОЛЬНИХ МАТЧІВ ТА ТУРНІРНИХ ТАБЛИЦЬ	21
2.1 Первинний аналіз даних та формування навчального датасету	21
2.2 Конструювання та оптимізація ознак для моделей прогнозування.....	32
2.3 Реалізація моделей прогнозування футбольних матчів та фінальної турнірної таблиці.....	42
Висновки до розділу 2	55
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІТИКИ ФУТБОЛЬНИХ МАТЧІВ	57
3.1 Розробка програмного рішення для практичного застосування моделей прогнозування.....	57
3.2 Опис користувацької взаємодії з розробленим програмним рішенням	62
Висновки до розділу 3	71
ВИСНОВКИ.....	73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	75
ДОДАТКИ.....	79

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface
CSV – Comma-Separated Values
GPU – Graphics Processing Unit
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
JSON – JavaScript Object Notation
LSTM – Long Short-Term Memory
MLS – Major League Soccer
MVC – Model-View-Controller
ORM – Object-Relational Mapping
ReLU – Rectified Linear Unit
RPS – Ranked Probability Score
SQL – Structured Query Language
TPU – Tensor Processing Units
XGBoost – Extreme Gradient Boosting
СУБД – Система управління базами даних

ВСТУП

Актуальність роботи. Прогнозування результатів футбольних матчів залишається однією з ключових задач сучасної спортивної аналітики. Складність цієї задачі зумовлена високою варіативністю гри, невеликою кількістю голів та впливом численних зовнішніх чинників, що робить оцінку ймовірностей перемоги, нічиєї чи точного рахунку нетривіальною задачею. Попри значні обсяги доступних даних, досі відсутній консенсус щодо оптимального підходу до моделювання, а порівняльні дослідження різних методів на однакових даних залишаються нечисленними.

Також важливою є задача прогнозування фінальної турнірної таблиці. Вона потребує врахування довгострокової динаміки сезону, послідовності результатів, календаря матчів та взаємного впливу команд протягом усього чемпіонату. На відміну від короткострокових прогнозів окремих поєдинків, цей напрям залишається менш дослідженим, хоча має високу практичну цінність для футбольних клубів, аналітиків і стратегічного планування.

Мета роботи полягає в створенні системи, яка прогнозує результат окремого футбольного матчу та ймовірність точного рахунку, а також визначає ймовірність підсумкового місця команди у фінальній турнірній таблиці.

Завданнями роботи є:

1. Проаналізувати доступні джерела футбольних даних та сформувати набір характеристик, необхідних для прогнозування матчів і турнірної таблиці.
2. Очистити та підготувати дані для подальшого моделювання.
3. Побудувати підсистему прогнозування результатів окремого матчу та оцінки ймовірності точного рахунку.
4. Розробити підсистему симуляції сезону для визначення ймовірних підсумкових місць команд у турнірній таблиці.
5. Об'єднати обидві підсистеми в єдину аналітичну систему та реалізувати її програмно.

6. Провести експериментальне дослідження роботи системи й оцінити точність прогнозів.

7. Проаналізувати отримані результати та визначити ефективність запропонованого підходу.

Об'єктом дослідження у магістерській роботі є футбольні матчі та турнірні чемпіонати як явище спортивної діяльності та їхні статистичні характеристики, що відображають результати поєдинків і підсумкові розташування команд у турнірних таблицях.

Предметом дослідження у магістерській роботі є методи та алгоритми прогнозування результатів футбольних матчів і формування ймовірного прогнозу фінальної турнірної таблиці на основі статистичних даних.

Методи дослідження у магістерській роботі включають збір та аналіз статистичних даних футбольних матчів, методи машинного навчання для прогнозування результатів поєдинків та фінальної турнірної таблиці, а також оцінку точності і ефективності розробленої системи.

Наукова новизна роботи полягає у проведенні порівняльного аналізу ефективності різних методів прогнозування футбольних матчів на єдиному наборі даних із застосуванням ймовірнісних метрик якості. Запропоновано підхід до прогнозування фінальної турнірної таблиці, який формує ймовірнісний розподіл підсумкових позицій для кожної команди ліги.

Практичне значення отриманих результатів магістерської роботи полягає у можливості використання розробленої системи для підтримки прийняття рішень у спортивних клубах і аналітичних відділах, планування тактики та стратегії команд, оцінки ймовірних результатів матчів та турнірних сценаріїв, а також для підготовки аналітичних звітів і прогнозів для спортивних медіа та букмекерських організацій.

Публікації. За темою атестаційної магістерської роботи було опубліковано тези [1-2].

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Прогнозування результатів футбольних матчів та формування фінальної турнірної таблиці є складним науково-технічним завданням, що вимагає системного підходу до аналізу даних та застосування сучасних методів моделювання. Основна складність полягає у необхідності врахування великої кількості змінних, що визначають результати команд у матчах і підсумкові позиції у турнірній таблиці. Традиційні статистичні показники, такі як забиті та пропущені голи, перемоги, поразки та нічиї, надають базове уявлення про результати команд, але не відображають повністю їхню динаміку протягом сезону. До числа додаткових чинників, що суттєво впливають на результати матчів належать форма команди, рівень суперників, календар матчів та інші зовнішні умови.

У межах дослідження передбачається проведення комплексного аналізу існуючих методів прогнозування з метою визначення найбільш ефективних підходів і їхньої адаптації для розробки системи аналітики футбольних матчів. У першу чергу здійснюється збір, структуризація та обробка великих масивів даних, що включають історичні та актуальні статистичні показники команд і гравців, результати матчів, а також додаткові характеристики, які можуть впливати на ігрові результати.

На підставі підготовлених даних розробляються алгоритми прогнозування окремих матчів, що забезпечують оцінку ймовірності перемоги, нічиєї та точного рахунку, із врахуванням особливостей команд і контекстних факторів. Паралельно реалізовуватиметься підсистема прогнозування підсумкових результатів сезону, яка дозволить на основі вже зіграної частини чемпіонату оцінювати подальший хід турніру. Використовуючи дані про обмежену кількість проведених матчів, система

формуватиме прогноз результатів решти зустрічей та визначатиме ймовірні підсумкові позиції команд у турнірній таблиці.

Особлива увага приділяється розробці релевантних характеристик команд, оптимізації алгоритмів та використанню сучасних методів машинного навчання для підвищення точності прогнозів. Інтеграція підсистем у єдину систему аналітики забезпечує можливість прогнозування різних сценаріїв розвитку турніру, аналізу впливу окремих факторів на результати матчів та підсумкові позиції команд, а також підготовки візуалізацій і аналітичних звітів для користувачів.

Завершальним етапом роботи є тестування системи на реальних даних футбольних ліг, оцінка точності прогнозів, аналіз помилок і вдосконалення алгоритмів для забезпечення високої надійності та практичної цінності розробленої системи. У результаті передбачається створення комплексної платформи, яка дозволить користувачам отримувати точні прогнози та змодельовану фінальну турніру таблицю.

1.2 Огляд наявних аналогів

Сучасна спортивна аналітика пропонує низку сервісів для прогнозування футбольних матчів і турнірних підсумків, серед яких особливою уваги заслуговують Opta Analyst та KickForm. Ці системи реалізують різні підходи до аналізу даних і прогнозування, що дозволяє окреслити їхні сильні і слабкі сторони та визначити можливості для використання у власній системі аналітики.

Система Opta Analyst забезпечує прогнозування як результатів окремих матчів, так і фінальної турнірної таблиці. Її інтерфейс можна побачити на рис. 1.1. В основі цієї системи лежить рейтинг команд Opta Power Rankings, який формується на основі історичних результатів і оцінки поточної форми команд. Для підвищення точності прогнозів конкретних матчів система інтегрує ринкові коефіцієнти букмекерів, що дозволяє враховувати зовнішні очікування. Значною перевагою Opta Analyst є поєднання короткострокового

і довгострокового прогнозу з використанням симуляцій сезону, що дає ймовірнісний розподіл підсумкових місць команд у таблиці. Водночас система має обмеження, зокрема залежність прогнозів від ринкових коефіцієнтів і недостатню прозорість внутрішніх алгоритмів оцінки команди [3].

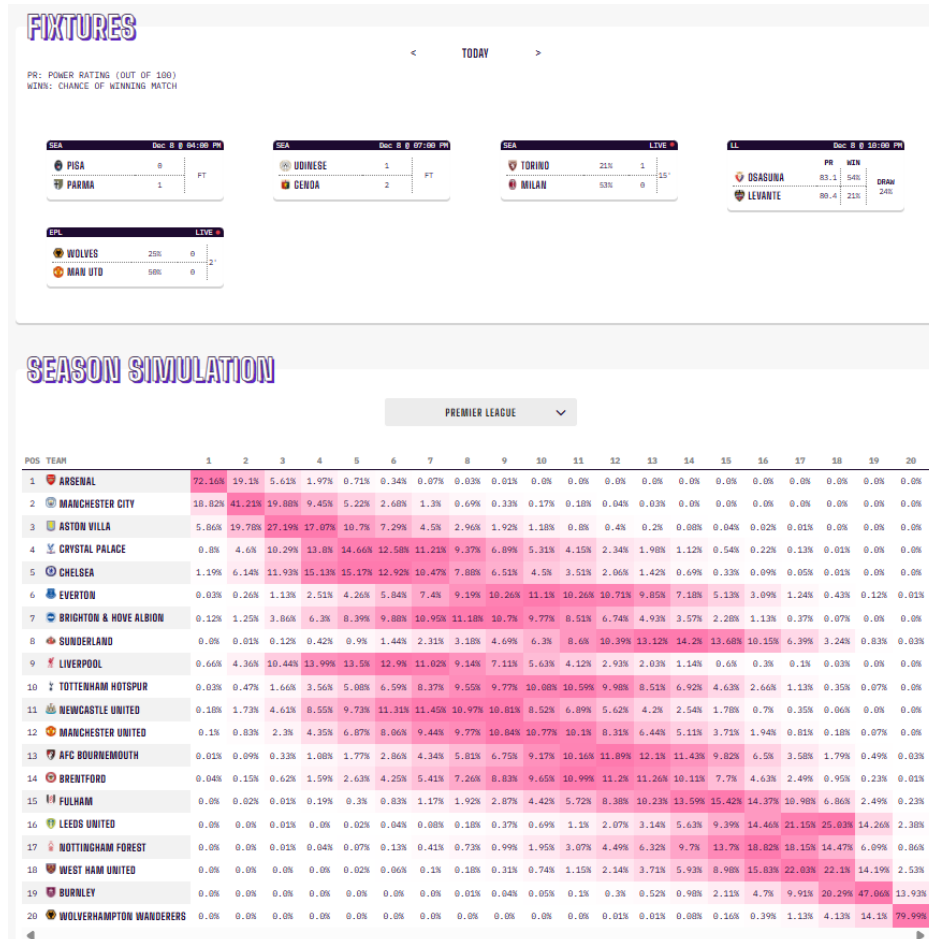


Рис. 1.1 – Інтерфейс сервісу Opta Analyst

Система KickForm, інтерфейс якої зображено на рис. 1.2, пропонує альтернативний підхід, орієнтований на оцінку потенціалу команд для прогнозу результатів окремих матчів. Алгоритм враховує домашню перевагу, форму команди і гравців, очікувані голяові моменти, ринкову вартість команди та інші фактори. Для визначення ймовірності точного рахунку застосовується моделювання за розподілом Пуассона. Перевагою KickForm є глибокий аналіз потенційних показників команд і можливість кастомізації під конкретну лігу або команду. Однак, як і у випадку з Opta Analyst, система має певну непрозорість алгоритму і орієнтована на ринок ставок, що зменшує академічну цінність для наукового аналізу [4].

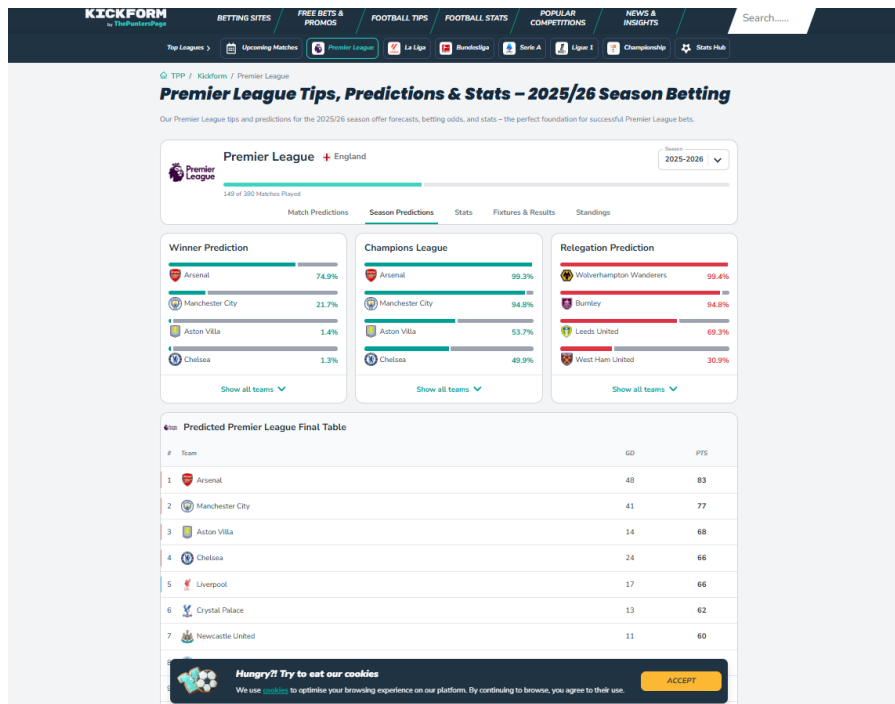


Рис. 1.2 – Інтерфейс сервісу KickForm

Для узагальнення характеристик обох сервісів та визначення їхньої придатності для розробки власної системи аналітики було сформовано порівняльну табл. 1.1. Вона дозволяє наочно оцінити типи прогнозів, методику, вхідні дані, сильні і слабкі сторони систем.

Таблиця 1.1

Порівняння характеристик розглянутих вебсервісів

Параметр / Система	Opta Analyst	KickForm
Тип прогнозу	Результати окремих матчів і симуляція турніру	Результати окремих матчів та прогноз ймовірності місць, підвищення/вильоту, Єврокубки
Методика	Рейтингові системи команд, симуляції сезону, інтеграція з ринковими коефіцієнтами	Оцінка голевих моментів, домашня перевага, форма, Poisson-моделювання точного рахунку
Вхідні дані	Історичні результати матчів, рейтинги команд, ринкові коефіцієнти	Історія голевих моментів, форма команд і гравців, домашня/гостьова статистика, ринкова вартість
Сильні сторони	Прогноз матчів і турніру, симуляції сезону, інтеграція даних і ринкових очікувань	Глибокий аналіз потенціалу команд, прогноз точного рахунку, можливість кастомізації
Слабкі сторони	Часткова залежність від ринкових коефіцієнтів, обмежена прозорість метрик	Орієнтація на ринок ставок, часткова непрозорість алгоритму, обмежена комплексність турнірного прогнозу

Таким чином, сучасні сервіси добре прогнозують як окремі матчі, так і підсумкові результати турніру. Але вони мають обмеження: алгоритми часто не прозорі, а самі системи більше орієнтовані на комерційні цілі. Щоб створити науково обґрунтовану систему, варто поєднати обидва підходи: використовувати симуляції для оцінки ймовірності місць команд у таблиці та аналізувати потенціал команд за статистикою. Водночас система має враховувати всі важливі характеристики команд і гравців і бути прозорою та гнучкою у використанні.

1.3 Вибір програмних засобів для реалізації

Розробка системи аналітики футбольних матчів є багатокомпонентним завданням, що вимагає ретельного вибору технологій для зберігання, обробки, аналізу та візуалізації даних. У процесі вибору програмних засобів було враховано вимоги до продуктивності, масштабованості та гнучкості, що дозволило сформулювати ефективний технологічний стек.

Для розроблення системи були обрані інструменти, які найкраще підходять для машинного навчання, роботи з даними, та створення вебзастосунку. Основою проєкту виступає мова програмування Python, оскільки вона має велику кількість бібліотек для аналізу даних, є простою у використанні й широко застосовується у сфері спортивної аналітики та штучного інтелекту.

Для обробки та підготовки даних використовуються бібліотеки NumPy та Pandas. NumPy забезпечує швидкі обчислення з масивами та матрицями, а Pandas дозволяє зручно працювати з табличними даними – очищати їх, фільтрувати, поєднувати та формувати вибірки для моделей.

Під час дослідження даних важливо бачити, як саме розподілені ознаки та як вони пов'язані між собою, тому для візуалізації використовується бібліотека Seaborn. Вона допомагає швидко будувати графіки, які полегшують аналіз та дозволяють знайти закономірності в даних.

Для створення моделей машинного навчання застосовується Scikit-Learn, одна з найпопулярніших бібліотек у цій сфері. Вона надає готові реалізації різних алгоритмів, зручні інструменти для тренування, тестування та оцінювання моделей. Це дозволяє експериментувати з різними методами й легко порівнювати їхню ефективність.

На етапах дослідження даних і побудови моделей використовувалося середовище Google Colaboratory. Це хмарна платформа для виконання Python-коду, яка надає доступ до обчислювальних ресурсів високої продуктивності, включно з GPU та TPU, і забезпечує повну відтворюваність експериментів завдяки збереженню всіх результатів і коду в хмарі.

Важливою складовою є вибір інструмента для створення серверної частини. У проєкті використано Flask – легкий вебфреймворк для Python, який дозволяє реалізувати MVC-подібну архітектуру. Flask надає механізми маршрутизації, обробки HTTP-запитів і рендерингу HTML-шаблонів за допомогою Jinja2, що робить його зручним для побудови вебінтерфейсу. Завдяки гнучкості, фреймворк дозволяє організувати застосунок у вигляді окремих модулів Blueprints, інтегрувати ORM, підключати сервіси прогнозування та створювати сторінки для відображення результатів моделі. Його мінімалістичний підхід дає змогу розробити власну структуру застосунку і робить Flask зручним інструментом для навчальних і дослідницьких проєктів.

Для роботи з базою даних використовується ORM-інструмент SQLAlchemy, який дозволяє описувати структуру таблиць у вигляді Python-класів та працювати з даними на більш високому рівні абстракції. Це спрощує управління збереженою інформацією та забезпечує незалежність від конкретної СУБД.

Додатково для організації архітектури застосунку використовується пакет Dependency Injector, який допомагає розділити логіку на окремі модулі та відповідає за впорядкування залежностей між ними. Це дозволяє створювати гнучку структуру, де кожен модуль інкапсулює свою

функціональність, а залежності між усіма частинами системи визначаються централізовано. А також такий підхід спрощує тестування і підтримку системи, оскільки є можливість змінювати одні модулі не зачіпаючи інші.

Для завантаження футбольних даних у проєкті буде використовуватися сервіс API-Football. Він дозволяє автоматично отримувати інформацію про матчі, команди, турнірні таблиці, статистику та розклади у зручному форматі JSON. Завдяки цьому дані завжди актуальні та їх легко підключати до процесу обробки та моделювання.

Разом ці технології утворюють повноцінний стек, який дозволяє пройти всі етапи, а саме від завантаження та обробки футбольних даних до побудови моделей машинного навчання та створення вебінтерфейсу.

1.4 Огляд наявних методів, моделей, алгоритмів прогнозування

У сфері спортивної аналітики для прогнозування результатів окремих футбольних матчів та формування підсумкових турнірних таблиць можуть застосовуватися різні моделі та алгоритми, які враховують специфіку гри, статистичні закономірності, якісний контекст і доступні дані. Оскільки будь-який аналіз чемпіонату починається саме з прогнозування результатів окремих матчів, передбачається доцільним спочатку розглянути ті методи, які можуть бути використані безпосередньо для формування ймовірнісних оцінок футбольних подій у межах одного матчу.

Одним із таких підходів є гібридна модель Пуассона, яка передбачає, що кількість забитих м'ячів окремою командою описується розподілом Пуассона. У цьому випадку вводяться параметри λ_{home} та λ_{away} , що відображають очікувану результативність господарів і гостей. Ці значення можуть визначатися регресійною моделлю на основі градієнтного бустингу, яка аналізує набір передматчевих ознак і формує оцінку математичного сподівання кількості голів (1.1, 1.2) [5]:

$$\lambda_{home} = E[\text{голи господарів}], \quad (1.1)$$

$$\lambda_{away} = E[\text{голи гостей}], \quad (1.2)$$

де $E[\text{голи}]$ – математичне сподівання, тобто середня очікувана кількість голів.

Після отримання параметрів передбачається, що кількість забитих м'ячів підпорядковується формулі розподілу Пуассона (1.3):

$$P(K = k | \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad (1.3)$$

де λ – параметр інтенсивності (очікувана кількість голів);

k – конкретна кількість забитих м'ячів;

e – основа натурального логарифму.

Модель ґрунтується на припущенні про незалежність кількості голів двох команд, тому ймовірність конкретного рахунку $h:a$ може описуватися добутком двох пуассонівських розподілів (1.4):

$$P(h, a) = P_{home}(h) \cdot P_{away}(a), \quad (1.4)$$

де h – кількість голів господарів;

a – кількість голів гостей.

Оскільки формується повна матриця ймовірностей усіх можливих рахунків, на її основі можуть обчислюватися більш загальні футбольні події. Наприклад, подія «обидві команди заб'ють» може описуватися як (1.5):

$$P(\text{обидві команди заб'ють}) = \sum_{h \geq 1} \sum_{a \geq 1} P(h, a), \quad (1.5)$$

Ймовірність того, що загальна кількість голів перевищить поріг T , може описуватися виразом (1.6) [6]:

$$P(\text{тотал понад } T) = \sum_{h=0}^{\infty} \sum_{a=0}^{\infty} 1(h + a > T)P(h, a), \quad (1.6)$$

де $1(h + a > T)$ – індикаторна функція, яка дорівнює 1, якщо сума $h + a$ більша за T , і 0 – в іншому випадку.

Таким чином, гібридний підхід поєднує математичну строгість пуассонівського розподілу з можливістю визначати параметри λ засобами машинного навчання.

Іншим варіантом моделювання результатів матчу є алгоритм випадкового лісу запропонований Л. Брейманом [7]. Це ансамблевий метод машинного навчання, який об'єднує прогнози багатьох дерев рішень. Основна ідея полягає в тому, що група незалежних дерев разом формує точніший прогноз, ніж кожне окремо – подібно до того, як колективна думка групи експертів часто виявляється кращою за думку одного спеціаліста.

Алгоритм працює наступним чином: спочатку будується велика кількість дерев рішень, кількість яких зазвичай становить від сотні до кількох тисяч, причому кожне дерево навчається на власній випадковій підвибірці даних. Кожне окреме дерево рішень є відносно простим класифікатором, який може бути схильним до перенавчання – надмірного пристосування до тренувальних даних. Однак коли багато таких дерев об'єднуються в ансамбль, їхні індивідуальні помилки усереднюються. Це робить випадковий ліс значно стійкішим та надійнішим порівняно з одним деревом рішень.

Фінальний прогноз формується голосуванням, де кожне дерево «голосує» за певний клас, а результатом стає клас із найбільшою кількістю голосів. Ймовірність події оцінюється як частка дерев, що проголосували за неї. Наприклад, якщо зі 100 дерев 45 проголосували за перемогу господарів, 30 за нічию і 25 за перемогу гостей, то модель прогнозує перемогу господарів з ймовірністю 0.45, нічию з ймовірністю 0.30 та перемогу гостей з ймовірністю 0.25.

Корисною властивістю алгоритму є можливість оцінити важливість ознак, адже під час побудови дерев модель випадкового лісу відстежує, наскільки кожна ознака покращує якість прогнозу. На основі цього можна визначити, які характеристики найбільше впливають на результат. Наприклад, чи поточна форма команди важливіша за історію особистих зустрічей. Така інформація корисна не лише для прогнозування, а й для розуміння того, що саме найбільше впливає на результати футбольних матчів.

Іншим напрямом є багатошарові штучні нейронні мережі. Вони здатні виявляти складні нелінійні взаємозв'язки між ознаками. У нейронній мережі дані проходять через кілька шарів обробки, де кожен шар формує власне представлення інформації. Припускається, що така структура може дозволяти моделі краще вловлювати приховані залежності, які не завжди помітні в класичних статистичних методах. Нейронна мережа може одночасно формувати прогнози різних футбольних подій, використовуючи єдине приховане представлення даних.

Одним з найефективніших методів роботи з послідовними даними є нейронні мережі LSTM. LSTM – це тип рекурентної нейронної мережі, яка може обробляти послідовні дані, зберігаючи важливі залежності навіть протягом тривалих періодів. Це робить її особливо корисною для завдань, що вимагають довгострокового врахування минулих результатів, таких як прогнозування підсумкового рейтингу футбольної ліги. Завдяки своїй здатності зберігати та аналізувати інформацію з попередніх матчів, мережі LSTM можуть відстежувати зміни та тенденції в результатах команд протягом сезону, що дозволяє більш точно прогнозувати їхні шанси на остаточне місце в турнірній таблиці [8].

У рекурентних нейронних мережах LSTM кожна комірка обробляє дані, що надходять з попереднього кроку, і зберігає важливу інформацію з попередніх кроків. Основна мета LSTM полягає у фільтрації непотрібної інформації, що досягається завдяки механізмам забуття, введення та виведення інформації. Усі вони мають спільну комірку. Ці комірки

називаються воротами. Щоб вирішити, яка інформація є важливою, а яка ні, LSTM використовують ці ворота, які по суті є нейронами з сигмоїдною функцією активації. Вони вирішують, яку частину інформації зберегти у відповідних секціях, ефективно діючи як воротарі, які пропускають через себе лише частину інформації. Використання сигмоїдної функції в цьому контексті є стратегічним, оскільки вона видає значення від 0 до 1, які безпосередньо відповідають часткам інформації, яку ми маємо намір зберегти. Наприклад, значення 1 означає, що вся інформація буде збережена, значення 0,5 означає, що буде збережена тільки половина інформації, а значення 0 означає, що вся інформація буде відкинута [9].

Повторювальний модуль в LSTM показаний на рис. 1.3 [10].

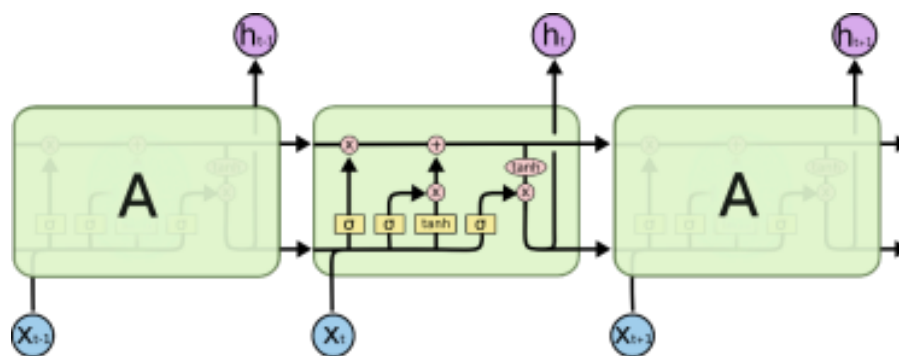


Рис. 1.3 – Повторюваний модуль в LSTM

Процес прогнозування фінальної таблиці за допомогою моделі LSTM базується на поетапному аналізі кожного матчу сезону. У цьому контексті етап означає обробку даних після кожного окремого матчу протягом футбольного сезону. На кожному етапі модель збирає новий набір статистичних даних про команду, включаючи забиті та пропущені голи, набрані очки, а також такі фактори, як зміни у складі або тактичних налаштуваннях. У кожній комірці модель оновлює свої векторні стани, зберігаючи важливу інформацію про форму команди. Довгострокова пам'ять LSTM дозволяє моделі зберігати ключові шаблони для довгострокового прогнозування, тоді як короткострокова пам'ять коригує поточні прогнози на основі останніх матчів. Це дозволяє моделі адаптуватися до коливань у результатах команди, таких як

серії перемог або поразок, і враховувати, як ці зміни можуть вплинути на загальний рейтинг у турнірній таблиці [11].

У результаті прогнозування результатів футбольних матчів та фінальних турнірних таблиць може здійснюватися різними підходами, кожен з яких має власні сильні сторони. Статистичні моделі забезпечують математичну основу для опису розподілів результатів, а машинне навчання дозволяє враховувати складні закономірності та велику кількість ознак.

Висновки до розділу 1

У розділі проаналізовано задачу прогнозування результатів футбольних матчів. Визначено основні складності: висока варіативність результатів, невелика кількість голів та вплив багатьох факторів.

Обрано технологічний стек: Python, Flask, SQLAlchemy, бібліотеки машинного навчання та API-Football для отримання даних.

Розглянуто три підходи до прогнозування: гібридна модель Пуассона для оцінки ймовірностей рахунків, алгоритм випадкового лісу та багат шарові нейронні мережі. Для прогнозування підсумкових турнірних таблиць обрано рекурентну нейронну мережу типу LSTM, яка враховує послідовність матчів у часі. Обрані методи будуть реалізовані та порівнянні у наступному розділі.

РОЗДІЛ 2. РОЗРОБКА МОДЕЛЕЙ ПРОГНОЗУВАННЯ РЕЗУЛЬТАТІВ ФУТБОЛЬНИХ МАТЧІВ ТА ТУРНІРНИХ ТАБЛИЦЬ

2.1 Первинний аналіз даних та формування навчального датасету

Дані, на основі яких здійснюється навчання моделей машинного навчання, визначають якість та коректність отриманих результатів. Архітектура моделі та вибір алгоритму не можуть компенсувати помилки у вихідних даних, їхню неповноту або некоректну часову узгодженість. Тому підготовка даних є базовим етапом, що передує побудові моделей прогнозування.

У межах цієї роботи для формування навчальної вибірки було використано відкритий набір даних з платформи Kaggle, який містить структуровану інформацію про футбольні матчі, команди, турнірні таблиці та статистичні показники гри [12]. Вибір саме цього набору даних обумовлений низкою технічних причин. Дані надаються у вигляді набору реляційних таблиць у форматі CSV. Таблиці логічно розділені за типами сутностей: матчі, команди, турнірні таблиці, статистика команд, склади та характеристики гравців. Така структура є зручною для аналізу та подальшої обробки. Наявність часових міток дозволяє відтворити хронологію подій, а обсяг вибірки є достатнім для тренування моделей машинного навчання. Окрім того, дані не є синтетичними – вони сформовані на основі реальних матчів, що забезпечує реалістичні розподіли ознак.

Базовою таблицею є таблиця матчів, яка визначає структуру майбутнього датасету. Один рядок таблиці відповідає одному матчу. Таблиця містить унікальний ідентифікатор матчу, дату та час його початку, ідентифікатори домашньої та гостьової команд, а також ідентифікатор ліги та тип сезону. Окремо зберігається рахунок матчу та бінарні індикатори перемоги кожної з команд. Поля з рахунком використовуються виключно як цільові змінні і не застосовуються під час формування ознак, що дозволяє

уникнути витоку даних. Окрім базової таблиці матчів, набір містить й інші таблиці з інформацією про виступи команд.

Турнірні таблиці зберігають агреговану інформацію про результати команд у межах сезону: позицію в таблиці, кількість перемог, нічий та поразок, набрані очки, забиті та пропущені голи, а також поточну форму у вигляді послідовності останніх результатів. Ці дані відображають довгострокову силу команди і можуть суттєво впливати на прогноз. Варто зазначити, що турнірні таблиці мають чітку часову залежність, тому при подальшому об'єднанні слід враховувати лише інформацію, доступну до моменту матчу.

Статистика матчів містить детальні кількісні показники гри: володіння м'ячем, кількість ударів та ударів у площину воріт, передачі, оборонні дії та дисциплінарні показники. Ці змінні потенційно корисні для побудови ознак ефективності та інтенсивності гри. Слід врахувати, що статистика доступна не для всіх матчів, тому пропущені значення потребуватимуть окремої обробки.

Таблиці зі складами містять інформацію про гравців, які брали участь у кожному матчі, включаючи стартовий склад та заміни. Окрема довідкова таблиця зберігає характеристики гравців: вік, зріст, вагу та ігрову позицію. На основі цих даних можна формувати агреговані ознаки складу команди, що дозволить оцінити фізичні характеристики команди перед матчем.

Набір також містить таблицю ключових подій матчу, де фіксуються голи, заміни, жовті та червоні картки з прив'язкою до конкретної хвилини гри. Ці дані можуть бути корисними для детального аналізу динаміки матчів та виявлення закономірностей у часі забитих голів.

Окремо зберігаються довідкові таблиці: інформація про ліги, команди та стадіони. Таблиця ліг містить назву турніру, країну та тип змагання. Таблиця команд зберігає повну та скорочену назви, а також кольори форми. Таблиця стадіонів включає назву, місто та місткість арени. Ці довідкові дані забезпечують контекст для кожного матчу та можуть бути використані для категоріальних ознак.

Повну структуру даних наведено на рис. 2.1.

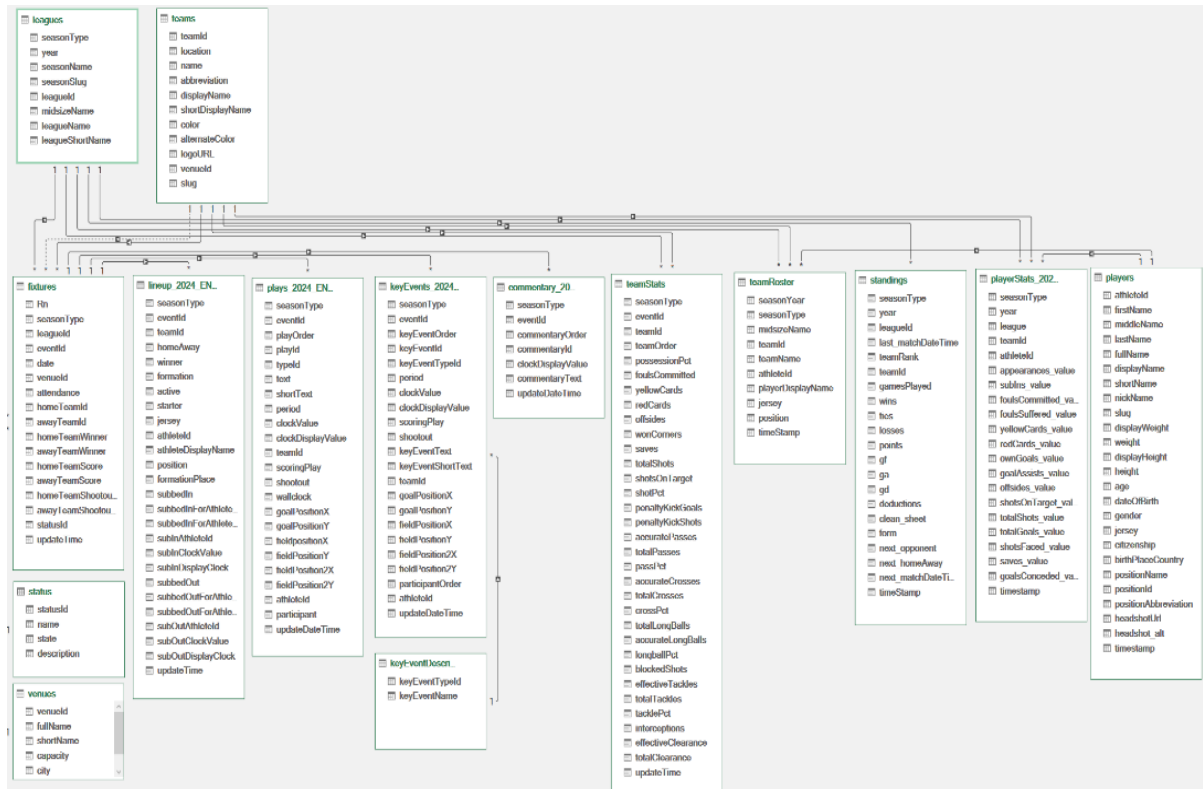


Рис. 2.1 – Повна структура набору даних

Після ознайомлення зі структурою даних, наступним кроком є розвідувальний аналіз. Він передусє побудові моделей і дозволяє краще зрозуміти природу даних. На цьому етапі дослідник вивчає розподіли ознак, виявляє закономірності та аномалії, перевіряє наявність пропущених значень та викидів.

У межах цієї роботи розвідувальний аналіз буде виконано з метою оцінки розподілів основних змінних та виявлення статистичних залежностей. Особлива увага буде приділено перевірці якості даних та виявленню можливих дисбалансів у цільових змінних, а результати аналізу допоможуть визначити стратегію подальшої обробки, зокрема, заповнення пропущених значень, нормалізацію змінних та створення нових ознак.

На першому етапі було проаналізовано розподіл кількості забитих голів домашніми та гостьовими командами, що представлено на рис. 2.2.

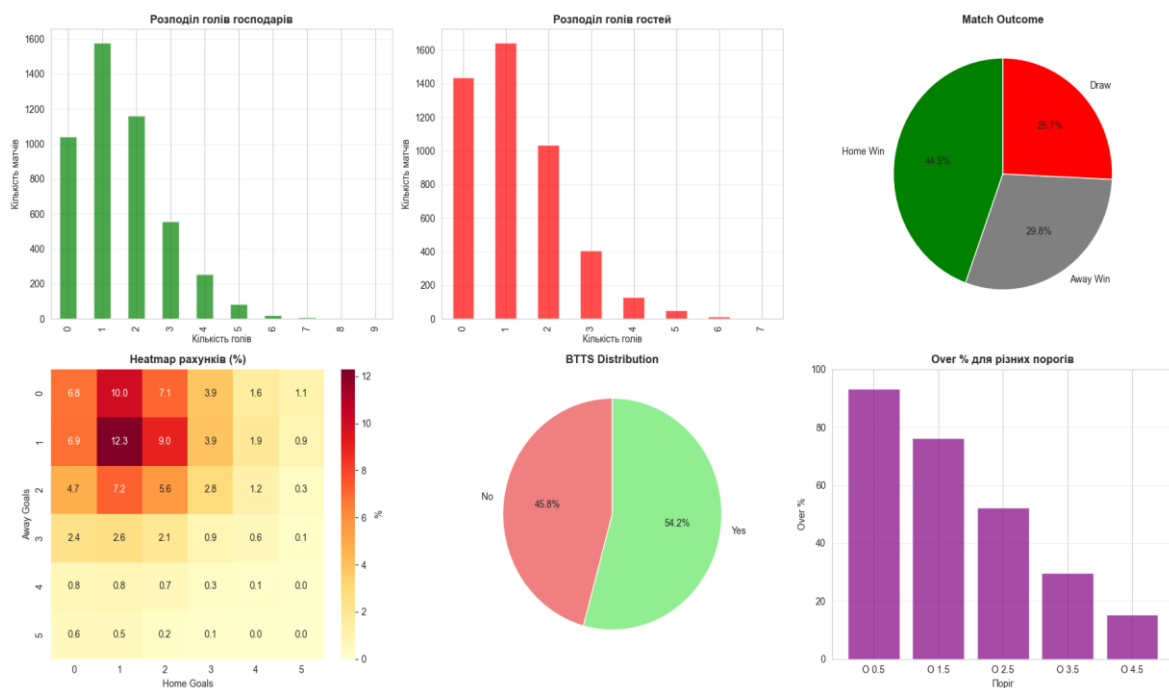


Рис. 2.2 – Аналіз забитих голів господарів та гостей

Гістограми голів показують, що для обох команд найчастіше фіксується 0, 1 або 2 голи за матч, з піком на 1 голі. Різниця полягає в тому, що гостеві команди частіше залишаються без забитих м'ячів, ніж домашні, що підтверджує ефект домашнього поля. Матчі, де одна команда забиває 4 і більше голів, трапляються вкрай рідко.

За результатами матчів господарі перемагають у 44.5% випадків, гості – у 29.8%, нічії складають 25.7%. Теплова карта рахунків демонструє, що найпоширенішими результатами є 1:1 (12.3%), 1:0 (10.0%) та 2:1 (9.0%), тобто більшість матчів завершуються з невеликою кількістю голів. Також можна відзначити, що у 54.2% матчів обидві команди забивають хоча б по голу. Аналіз сумарної кількості голів показує, що понад 2.5 голів за матч забивається у 55% випадків, понад 3.5 – у 30%, а понад 4.5 – лише у 15%.

На рис. 2.3. представлено порівняльний аналіз національних ліг за чотирма показниками: середня кількість голів, домашня перевага (у різниці голів), відсоток матчів, де обидві команди забили та відсоток матчів з понад 2.5 голами.

За середньою кількістю голів лідирують MLS (3.15) та Бундесліга (3.13), тоді як аргентинська ліга має найнижчий показник (2.03). Загальне середнє по всіх лігах становить близько 2.7 голів.

Аналіз домашньої переваги показує, що в усіх лігах господарі мають перевагу, але її величина суттєво відрізняється. Найбільший ефект домашнього поля зафіксовано у мексиканській лізі (0.54) та бельгійській лізі (0.50), а найменший – в англійській Прем'єр-Лізі (0.09) та італійській Серії А (0.12). За показниками результативності MLS також лідирує: 63.7% матчів з голами обох команд та 62.3% матчів з понад 2.5 голами. Аргентинська ліга, навпаки, має найнижчі значення за всіма показниками (41.5% та 29.9% відповідно). Такі відмінності між лігами свідчать про доцільність врахування контексту ліги при формуванні ознак та навчанні моделей прогнозування.

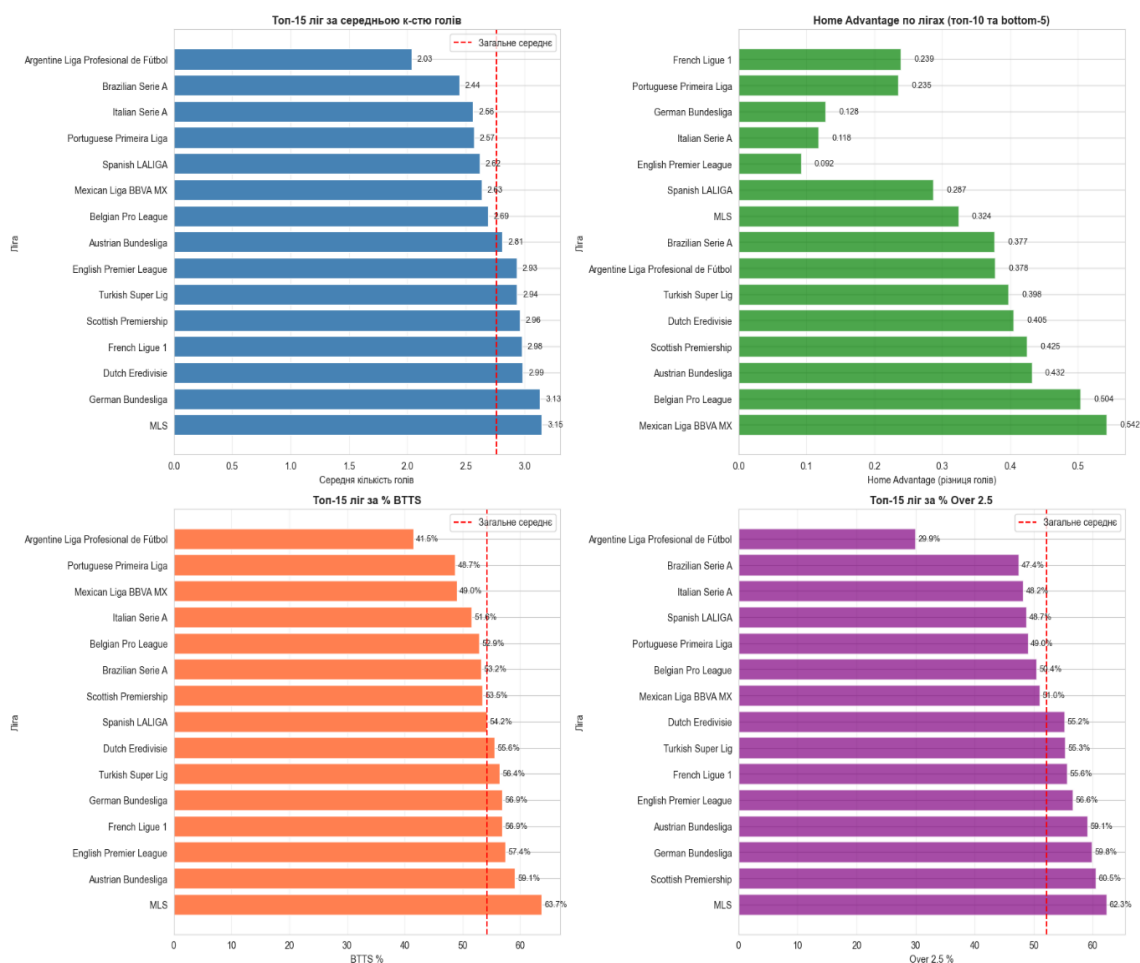


Рис. 2.3 – Аналіз домашньої переваги в різних національних лігах

На рис. 2.4 представлено залежність результатів матчів від позиції команди в турнірній таблиці.

Порівняння команд з верхньої та нижньої частин таблиці демонструє суттєву різницю у результатах. Команди з топ-3 перемагають приблизно у 55% матчів, тоді як команди з останньої трійки – лише у 24%. Ситуація з поразками є дзеркальною: для лідерів цей показник становить 24%, для аутсайдерів – 52%. Відсоток нічийх для обох груп залишається приблизно однаковим.

Аналіз залежності відсотка перемог від позиції показує чітку монотонну тенденцію: чим вища позиція команди, тим частіше вона перемагає. Команди-лідери мають відсоток перемог близько 60%, тоді як команди внизу таблиці – лише 20%. Ці результати свідчать про те, що позиція в турнірній таблиці є важливою ознакою для прогнозування результатів матчів.

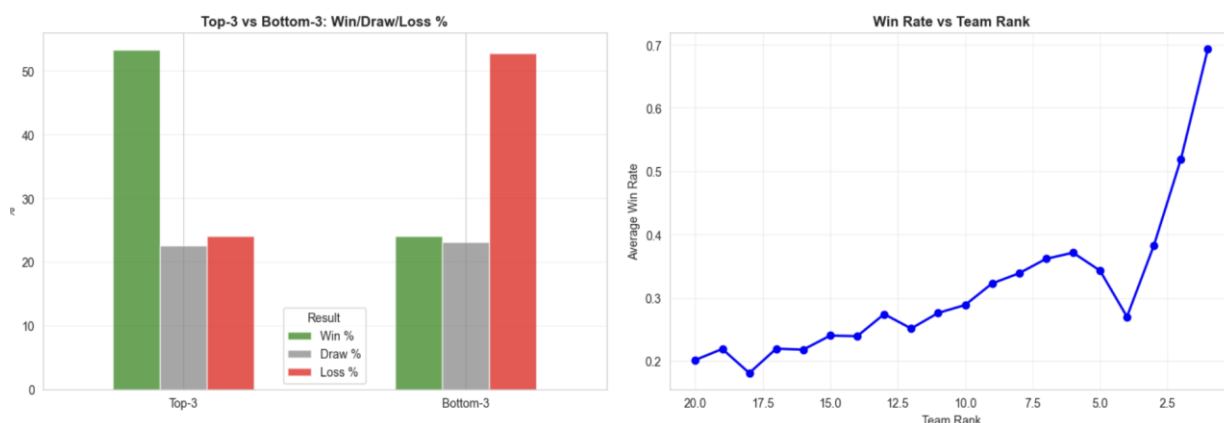


Рис. 2.4 – Залежність результатів від позиції команди в турнірній таблиці

Аналіз статистики ударів за матч, який зображено на рис. 2.5, показує, що в середньому команда наносить 12–13 ударів за матч, з яких близько 4 влучають у площину воріт. Це означає, що приблизно кожен третій удар є точним – середня точність становить 36%.

Між кількістю ударів та влучаннями у площину воріт існує очікуваний зв'язок: команди, які б'ють частіше, частіше і влучають у ціль. Однак при великій кількості ударів їхня якість зазвичай падає – команда починає бити з гірших позицій. Матчі з екстремальними показниками, такими як більше 30 ударів або точність понад 80%, трапляються рідко.

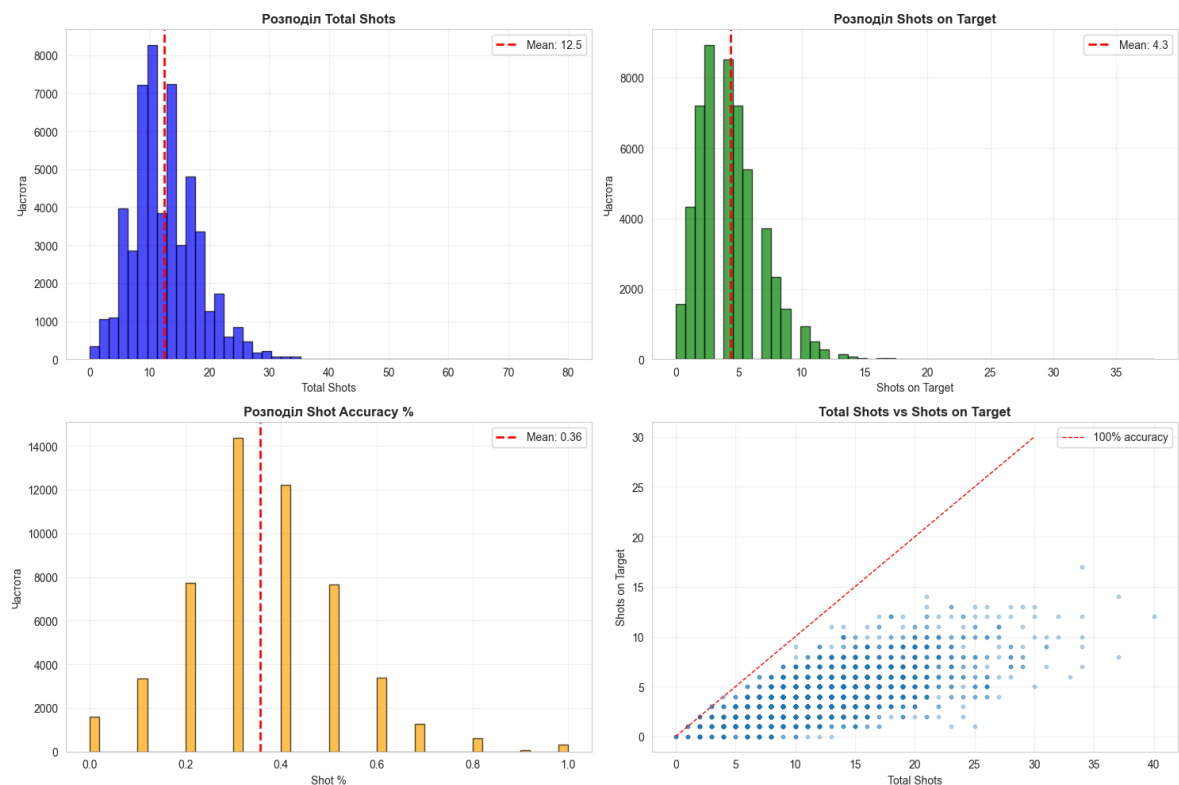


Рис. 2.5 – Аналіз статистики ударів за матч

Оборонна статистика команд, що зображена на рис. 2.6, демонструє типові показники захисних дій. Воротарі в середньому роблять 3 сейви за матч, при цьому найчастіше фіксується 2–3 сейви. Матчі з великою кількістю сейвів (10 і більше) трапляються рідко.

Команди виконують у середньому 16 відборів м'яча за гру – це ситуації, коли гравець вступає в єдиноборство із суперником і намагається забрати м'яч. Успішність таких дій становить близько 64%, тобто приблизно дві третини спроб є вдалими. Більшість команд демонструє успішність відборів у межах 50–80%.

Перехоплення команда здійснює в середньому 8–9 за матч. На відміну від відборів, перехоплення – це переривання передачі суперника, коли гравець «читає» гру і забирає м'яч без прямого єдиноборства. Розподіл цього показника є досить рівномірним у діапазоні 5–12 перехоплень, що свідчить про варіативність залежно від стилю гри та суперника.

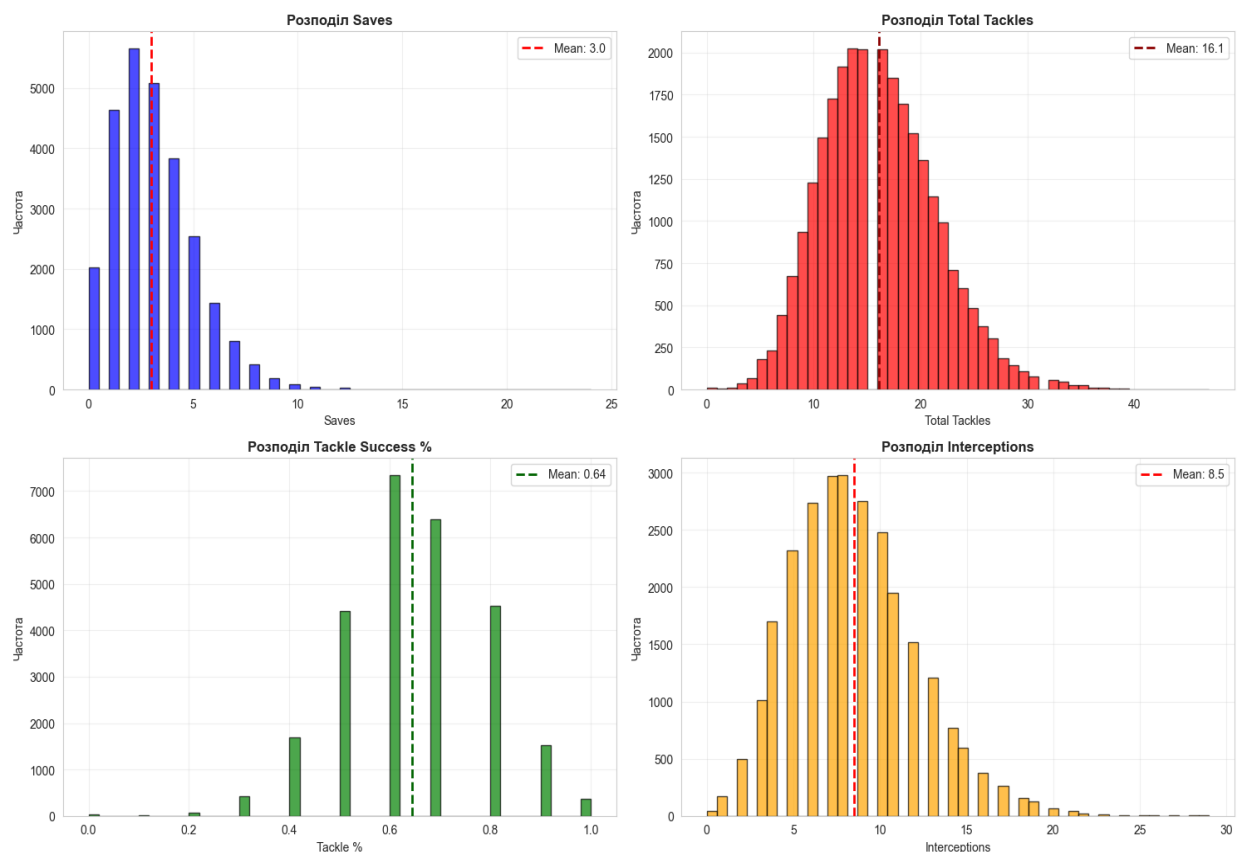


Рис. 2.6 – Аналіз оборонної статистики команд

Розвідувальний аналіз даних підтвердив коректність та внутрішню узгодженість використаного датасету. Було виявлено очікувані статистичні закономірності, характерні для футбольних матчів: ефект домашнього поля, залежність результатів від позиції в турнірній таблиці, типові розподіли голів та ігрової статистики.

Наступним кроком є об'єднання даних та формування єдиної структури. Основна задача полягає у трансформації розрізнених реляційних таблиць у єдину пласку структуру, де кожен рядок відповідатиме одному матчу з усіма необхідними ознаками.

Ключовою вимогою до процесу об'єднання даних є уникнення витоку інформації з майбутнього. Для цього приєднання всіх допоміжних таблиць здійснюється з урахуванням часової компоненти. У рядок з матчем включається виключно інформація, яка була доступна до дати проведення матчу.

Процес об'єднання починається з таблиці матчів, яка задає каркас майбутнього датасету. Вона містить ідентифікатори команд, дату матчу, ідентифікатор ліги та сезону. Ці поля використовуються як ключі для приєднання інших таблиць. Кількість рядків у фінальній таблиці дорівнює кількості матчів і не змінюється в процесі об'єднання.

До кожного матчу також приєднуються дані з турнірних таблиць, такі як позиція команди, кількість очок, кількість зіграних матчів, забиті та пропущені голи. Оскільки турнірна таблиця постійно оновлюється після кожного матчу, важливо використовувати лише ту інформацію, яка була актуальна до початку гри. Для цього застосовується функція `merge_asof` з напрямком `backward`, яка для кожного матчу знаходить останній запис турнірної таблиці, дата якого не перевищує дату матчу. Об'єднання виконується окремо для домашньої та гостьової команд.

```
merged_home = pd.merge_asof(
    fixtures_sorted,
    standings_sorted,
    left_on='date',
    right_on='last_matchDateTime',
    by=['homeTeamId', 'leagueId'],
    direction='backward',
    suffixes=('', '_home')
)
```

Перед об'єднанням з таблицею проміжних турнірних результатів виконується конвертація дат та видалення надлишкових колонок. З таблиці було обрано ключові показники: позиція команди, набрані очки, забиті та пропущені голи, матчі без пропущених голів та поточна форма.

За аналогією виконується об'єднання зі статистикою команд у матчах. Ця таблиця містить детальні показники гри: володіння м'ячем, кількість ударів, передач, відборів та дисциплінарні показники.

Аналогічно приєднуються дані про склади команд та характеристики гравців. На основі стартового складу обчислюються середній вік та зріст команди. Ці характеристики можуть впливати на стиль гри та результативність команди.

Після об'єднання інформації з різних джерел у сформованому датасеті виникає значна кількість пропущених значень, що пов'язано з неповним покриттям окремих таблиць, часовими обмеженнями при об'єднанні даних, а також особливостями розрахунку похідних показників. Без попереднього заповнення таких пропусків використання більшості алгоритмів машинного навчання є неможливим.

У цій роботі свідомо не застосовується видалення рядків з пропущеними значеннями. Кожен рядок датасету відповідає окремому футбольному матчу, тому його виключення призводило б до втрати цінної інформації та зменшення репрезентативності вибірки. Натомість використовується системна стратегія імпутації, яка враховує семантику ознак, контекст команди та рівень ліги.

Основними джерелами пропусків є таблиці детальної матчевої статистики, інформація про склади команд та індивідуальні показники гравців, які доступні лише для частини матчів. Додатково пропуски виникають на початку сезонів, коли відсутня історія для розрахунку агрегованих показників.

Для заповнення відсутніх значень було застосовано ієрархічну стратегію, яка використовує інформацію від більш специфічної до більш загальної. На першому рівні застосовуються характеристики конкретної команди. На другому рівні враховуються загальні тенденції у лізі, а на третьому рівні застосовується глобальна медіана по всьому датасету. Також, якщо повністю відсутні дані про володіння м'ячем командою, логічним є припущення рівного розподілу, тобто 50% на 50%.

Категоріальні дані заповнюються або тим значенням, яке найчастіше зустрічається, або спеціальним маркером, якщо інформація відсутня. Наприклад, зона клімату стадіону позначається як `unknown`, якщо координати стадіону відсутні. Технічно це реалізується наступним чином:

```
for col in categorical_cols:

    if df[col].isnull().any():
        mode_val = df[col].mode()
        df[col] = df[col].fillna(mode_val[0] if len(mode_val) > 0 else
                                'unknown')
```

Для ознак історії зустрічей команд відсутність попередніх матчів заповнюється нейтральними значеннями, які враховують форму команди в останніх матчах. Якщо команди не грали між собою, ймовірності перемоги домашньої та гостьової команди або нічиєї оцінюються як поєднання показників останніх п'яти матчів кожної команди, з урахуванням фактору домашнього поля:

```
h2h_defaults = {
    'h2h_home_win_pct': 0.55 * df['form_last_5_home'] + 0.45 *
df['form_last_5_away'],
    'h2h_draw_pct': 0.33,
    'h2h_away_win_pct': 0.45 * df['form_last_5_away'] + 0.55 *
df['form_last_5_home'],
    'h2h_avg_total_goals': df['total_goals'].median(),
}
```

Заповнення статистичних показників команд проводиться з урахуванням сили суперника, домашнього або гостьового статусу та поточного сезону. Спершу пропуски заповнюються медіаною команди в поточному сезоні, потім з урахуванням середньої сили суперника, після цього медіаною по чемпіонату, а у разі відсутності всіх цих значень застосовується глобальна медіана:

```
team_median = df.groupby([team_col,
    'seasonType'])[col].transform('median')
df[col] = df[col].fillna(team_median)
df[col] = df[col].fillna(df.groupby([team_col, home_transfer_value_tm
    '])[col].transform('median'))
league_median = df.groupby('leagueId')[col].transform('median')
df[col] = df[col].fillna(league_median)
df[col] = df[col].fillna(df[col].median())
```

Ознака очікуваних голів була введена у вигляді наближеного показника, обчисленого на основі кількості ударів по воротах, володіння м'ячем та поточної форми команди. У разі відсутності цього показника значення заповнюються медіаною відповідної ліги, а за її відсутності – глобальною медіаною.

```
proxy_col = 'home_shots_on_target' if 'home' in col else
    'away_shots_on_target'
df[col] = df[col].fillna(0.6 * df[proxy_col] + 0.4 *
    (df[possession_col]/100) * 3)
df[col] = df[col].fillna(df.groupby('leagueId')[col].transform('median'))
```

```
df[col] = df[col].fillna(df[col].median())
```

Отже, після виконання ієрархічної імпутації пропусків датасет стає повним і готовим до подальшої обробки. Усі відсутні значення були заповнені з урахуванням контексту команди, сили суперника, особливостей ліги та глобальних тенденцій. Категоріальні ознаки отримали логічні заміщення, числові показники були відновлені за допомогою медіан на різних рівнях.

2.2 Конструювання та оптимізація ознак для моделей прогнозування

Наступним етапом підготовки даних є конструювання ознак, зокрема процес створення нових змінних на основі існуючих даних, які краще відображають закономірності в даних та підвищують прогностичну здатність моделей. Первинні змінні вже є формально придатними для використання в алгоритмах машинного навчання, однак вони не забезпечують достатньо глибокого та структурованого опису футбольного матчу. Конструювання ознак спрямоване на розширення та збагачення початкового представлення матчу шляхом формування похідних показників.

Однією з базових груп ознак є показники результативності команд у перерахунку на матч. Абсолютні значення, такі як загальна кількість забитих голів або набраних очок, залежать від кількості зіграних матчів і тому не є напряду порівнюваними. Для усунення цього ефекту використовуються нормалізовані метрики. Середня кількість забитих голів за матч обчислюється за формулою, де чисельник відповідає загальній кількості забитих голів команди, а знаменник – кількості зіграних матчів на момент прогнозу. Ця ознака відображає базову атакувальну силу команди незалежно від тривалості сезону. Ця формула матиме наступний вигляд (2.1) [13]:

$$GoalsPerGame = \frac{GoalsFor}{GamesPlayed}, \quad (2.1)$$

Аналогічним чином формується показник середньої кількості пропущених голів за матч. Цей показник використовується як оцінка оборонної стабільності команди. Менші значення свідчать про надійнішу гру в захисті (2.2):

$$GoalsAgainstPerGame = \frac{GoalsAgainst}{GamesPlayed}, \quad (2.2)$$

Для характеристики загального успіху команди використовується кількість очок у перерахунку на матч (2.3):

$$PointsPerGame = \frac{Points}{GamesPlayed}, \quad (2.3)$$

Окрему групу складають ознаки ефективності, що будуються на основі детальної статистики матчів. Наприклад, точність ударів визначається як відношення ударів у площину воріт до загальної кількості ударів. Цей показник характеризує якість створених моментів, а не лише їхню кількість й записується таким чином (2.4) [14]:

$$ShotAccuracy = \frac{ShotsOnTarget}{TotalShots+\varepsilon}, \quad (2.4)$$

де ε – мала константа, що запобігає діленню на нуль.

Аналогічно розраховується точність передач. Високі значення цієї ознаки вказують на здатність команди контролювати м'яч і темп гри (2.5):

$$PassAccuracy = \frac{AccuratePasses}{TotalPasses+\varepsilon}, \quad (2.5)$$

Для опису інтенсивності атак використовується кількість ударів на одиницю володіння м'ячем. Цей показник дозволяє розрізняти команди, які

контролюють м'яч без створення моментів, і команди, що грають більш вертикально та ефективно (2.6):

$$ShotsPerPossession = \frac{TotalShots}{PossessionPercentage + \varepsilon}, \quad (2.6)$$

Важливе місце в конструюванні ознак займають характеристики індивідуальної якості гравців. Для цього використовується інформація про стартовий склад. Середній вік стартового складу визначається за наступною формулою. Ця ознака відображає баланс між досвідом та фізичними можливостями команди (2.7) [15]:

$$AvgAge = \frac{1}{11} \sum_{i=1}^{11} Age_i, \quad (2.7)$$

де i – індекс гравця у стартовому складі, який змінюється від 1 до 11 відповідно до кількості футболістів, що одночасно перебувають на полі.

Фізичні характеристики команди описуються середнім зростом та вагою гравців (2.8, 2.9):

$$AvgHeight = \frac{1}{11} \sum_{i=1}^{11} Height_i, \quad (2.8)$$

$$AvgWeight = \frac{1}{11} \sum_{i=1}^{11} Weight_i, \quad (2.9)$$

Якість воротаря оцінюється через відносну кількість відбитих ударів. Цей показник є нормалізованою оцінкою ефективності гри воротаря та має значення в діапазоні від 0 до 1 (2.10):

$$GoalkeeperSaveRate = 1 - \frac{GoalsConceded}{ShotsOnTargetAgainst + \varepsilon}, \quad (2.10)$$

Показник результативності нападників визначається як відношення загальної кількості забитих м'ячів нападником до кількості матчів, у яких він брав участь. У чисельнику формули використовується загальна кількість голів, що відображає сумарний результат атакуювальних дій гравця за розглянутий період. У знаменнику використовується кількість матчів участі, тобто число ігор, у яких футболіст виходив на поле незалежно від тривалості ігрового часу (2.11).

$$GoalsPerAppearance_{FWD} = \frac{TotalGoals_{FWD}}{Appearance_{FWD}}, \quad (2.11)$$

Крім показника результативності, використовується коефіцієнт реалізації моментів, який визначається як відношення кількості забитих м'ячів до загальної кількості ударів по воротах (2.12) [16].

$$ConversioneRate = \frac{Goals}{TotalShots + \varepsilon}, \quad (2.12)$$

Для оцінки захисної сили команди використовується показник, який порівнює кількість відборів м'яча та перехоплень із кількістю атак суперника та часом, коли команда грає без м'яча. Такий підхід дозволяє врахувати, що сильні команди обороняються рідше, але це не означає слабку оборону, і дає більш об'єктивну оцінку захисних дій (2.13).

$$DefensiveIntensity = \frac{\frac{(Tackles + Interceptions)}{GamesPlayed}}{\frac{(OpponentShots)}{GamesPlayed} \cdot (1 - Possession) + \varepsilon}, \quad (2.13)$$

Окрім довгострокових турнірних показників, важливу роль у прогнозуванні результатів футбольних матчів відіграє поточна форма команд. На відміну від агрегованих сезонних метрик, форма відображає короткострокову динаміку виступів та дозволяє врахувати тимчасові зміни у

грі команди, пов'язані з кадровими ротаціями, травмами, календарним навантаженням або змінами тактичних підходів.

Форма команди моделюється на основі результатів її попередніх матчів. Для кількісного представлення результатів кожному матчу присвоюється числове значення: перемога інтерпретується як 3 бали, нічия – як 1 бал, поразка – як 0 балів. Числове значення форми команди за останні N матчів визначається як середнє арифметичне отриманих балів. Значення цієї ознаки знаходиться в діапазоні від 0 до 3 та відображає середню кількість очок, які команда набирає за матч у короткостроковій перспективі (2.14).

$$\text{FormScore}_N = \frac{1}{N} \sum_{i=1}^N \text{Result}_i, \quad (2.14)$$

де N – кількість попередніх матчів, що враховуються під час оцінювання форми команди.

Для кожного матчу окремо обчислюються показники форми домашньої та гостьової команд. Оскільки модель прогнозує результат конкретного протистояння, ключовою є не абсолютна форма команд, а їхній відносний стан. Тому додатково формується диференціальна ознака. Позитивне значення цієї ознаки означає, що домашня команда перебуває у кращій формі за суперника, тоді як від'ємне значення свідчить про перевагу гостьової команди (2.15) [17].

$$\text{FormDiff}_N = \text{FormScore}_{N,home} - \text{FormScore}_{N,away}, \quad (2.15)$$

Для оцінки поточної сили футбольних команд у роботі використано рейтинг Elo – динамічну числову метрику, яка оновлюється після кожного матчу з урахуванням фактичного результату та сили суперника [18].

На початку всі команди ініціалізуються однаковим базовим рейтингом 1500, після чого їхнє значення змінюються виключно на основі зіграних матчів.

Перед кожним матчем зчитуються поточні рейтинги команд, при цьому для домашньої команди додається фіксована поправка на перевагу домашнього поля. На основі різниці рейтингів обчислюється очікуваний результат матчу, після чого рейтинг команд оновлюється залежно від відхилення фактичного результату від очікуваного. Якщо команда виступає краще, ніж прогнозувалося, її рейтинг зростає, і навпаки.

Для більш глибокого опису потенціалу атаки команд у межах даної роботи використовується показник очікуваних голів. На відміну від фактичної кількості забитих м'ячів, показник очікуваних голів дозволяє оцінити якість створених голевих моментів незалежно від випадкових факторів, таких як індивідуальна реалізація або дії воротаря. Це робить xG корисним інструментом для моделювання сили атаки команди.

У роботі використовується наближена оцінка xG, оскільки детальні дані про кожен удар доступні не для всіх матчів. Для цього застосовується проксі-метрика, побудована на основі агрегованої матчевої статистики ударів. Очікувані голи команди оцінюються з урахуванням кількості ударів у площину воріт, точності завершення атак та ударів з пенальті (2.16) [19]:

$$xG_{proxy} = shotsOnTarget \cdot shot_{accuracy} \cdot \alpha + penaltyKickShots \cdot \beta, \quad (2.16)$$

де α – коефіцієнт, який показує, яка частина ударів у площину воріт у середньому завершується голом;

β – коефіцієнт, що характеризує середню результативність ударів з пенальті.

Для масштабування внеску різних типів моментів використовуються фіксовані коефіцієнти $\alpha=0.30$ та $\beta=0.75$. Значення α відповідає середній конверсії ударів у площину воріт у забиті голи, яка становить приблизно 30%, тоді як коефіцієнт β відображає середню реалізацію пенальті на рівні близько 75%.

Оскільки значення ознака очікуваних голів формується на основі матчевої статистики, яка стає відомою лише після завершення гри, безпосереднє використання цього показника для прогнозування поточного матчу є некоректним. Тому у фінальному датасеті для кожного матчу обчислюється середнє значення команди за попередні N матчів, що дозволяє оцінити поточну атакуючу форму команди [20].

Для оцінки фізичного стану команд у моделі враховується час відновлення між матчами та щільність календаря. Кількість днів відпочинку після попередньої гри дозволяє наближено оцінити рівень втоми гравців перед матчем, оскільки короткі інтервали між іграми часто призводять до накопичення втоми, ротації складу та зниження інтенсивності гри.

Отже, в результаті конструювання ознак було доповнено сформований датасет статистичними, рейтинговими та контекстні характеристиками футбольних матчів. Окрім ознак, детально розглянутих, до датасету також включено додаткові похідні змінні, зокрема агреговані показники трансферної вартості складів. Сукупність сформованих змінних забезпечує повний опис матчу та є достатньою для подальшого навчання моделей машинного навчання.

Після завершення формування набору даних та конструювання ознак, наступним етапом підготовки даних є відбір найбільш інформативних змінних. Сформований датасет містить велику кількість ознак, частина з яких може бути надлишковою або мати незначний вплив на якість прогнозу. Використання повного набору ознак без попереднього відбору ускладнює модель та може негативно впливати на її узагальнювальну здатність.

Відбір ознак виконується після завершення всіх етапів об'єднання, інженерії ознак та імпутації пропущених значень. На цьому етапі всі змінні мають коректний числовий або категоріальний формат і можуть бути використані в алгоритмах машинного навчання. Основною метою є зменшення розмірності вхідного простору при збереженні ключової інформації, необхідної для прогнозування результатів матчів.

На початковому кроці з датасету були виключені всі цільові змінні, а також похідні поля, які безпосередньо описують результат матчу. До них належать фінальний рахунок, бінарні індикатори перемоги, нічиєї та поразки. Такий підхід є обов'язковим для уникнення витоків інформації та забезпечення подальшого коректного навчання моделей прогнозування.

Для відбору ознак у роботі використано модель градієнтного бустингу над деревами рішень XGBoost у конфігурації MultiOutput. Такий підхід дозволяє одночасно моделювати декілька цільових змінних, зберігаючи єдиний простір вхідних ознак. У нашому випадку як цільові змінні розглядаються кількість забитих м'ячів командою та кількість забитих м'ячів суперником, що є доцільним для задач прогнозування футбольних матчів, оскільки ці показники безпосередньо визначають підсумковий результат гри, точні рахунки та загальну кількість голів [21].

XGBoost навчається поетапно. Спочатку модель будує дуже простий початковий прогноз, який ще не враховує всі особливості даних. Далі для кожного прикладу порівнюється цей прогноз із реальним значенням і визначається, наскільки і в який бік модель помилилася. На основі цих помилок будується наступне дерево рішень, яке спеціально націлене на ті ситуації, де попередній прогноз був неточним. Це дерево не замінює попередні, а додає поправку до вже отриманого прогнозу. У результаті кожен новий крок навчання зменшує загальну помилку моделі, поступово роблячи прогноз ближчим до реальних значень [22].

Використання дерев рішень дозволяє моделі самостійно виявляти складні залежності між ознаками та враховувати їхній спільний вплив без задання цих зв'язків вручну. Це є важливим для футбольних даних, оскільки багато факторів впливають на результат матчу не окремо, а лише в поєднанні з іншими ігровими характеристиками.

На рис. 2.7 наведено результати аналізу впливу кількості використаних ознак на якість прогнозування та рівень перенавчання моделі. Показано, що зі збільшенням кількості ознак помилка на навчальній вибірці поступово

зменшується, що свідчить про зростання здатності моделі відтворювати навчальні дані. Водночас значення помилки на валідаційній вибірці після досягнення певної кількості ознак перестає суттєво знижуватися та починає коливатися, що вказує на обмежений приріст узагальнювальної здатності моделі.

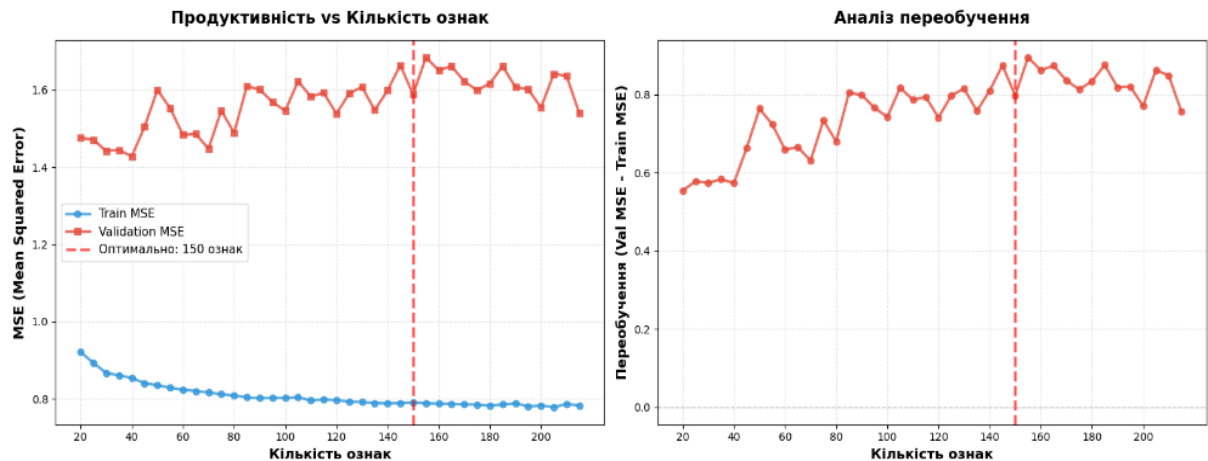


Рис. 2.7 – Аналіз впливу кількості ознак на якість моделі

Отримані результати свідчать, що використання приблизно сто п'ятдесят ознак є оптимальним з точки зору балансу між точністю прогнозування та узагальнювальною здатністю моделі. До цього рівня збільшення кількості ознак супроводжується зниженням помилки на валідаційній вибірці або її стабілізацією, що означає покращення якості прогнозу на нових даних. Після перевищення цієї кількості подальше розширення набору ознак уже не приводить до суттєвого покращення результатів на валідаційних даних, тоді як різниця між помилками на навчальній та валідаційній вибірках починає зростати.

На рис. 2.8 наведено розподіл важливості п'ятдесяти найбільш інформативних ознак, визначених за результатами навчання моделі XGBoost. Найбільший внесок у формування прогнозу мають показники, що безпосередньо відображають поточну силу та результативність команд. Зокрема, високою важливістю характеризуються ознаки, пов'язані з кількістю забитих і пропущених голів за матч, середньою кількістю очок, різницею рейтингів команд.

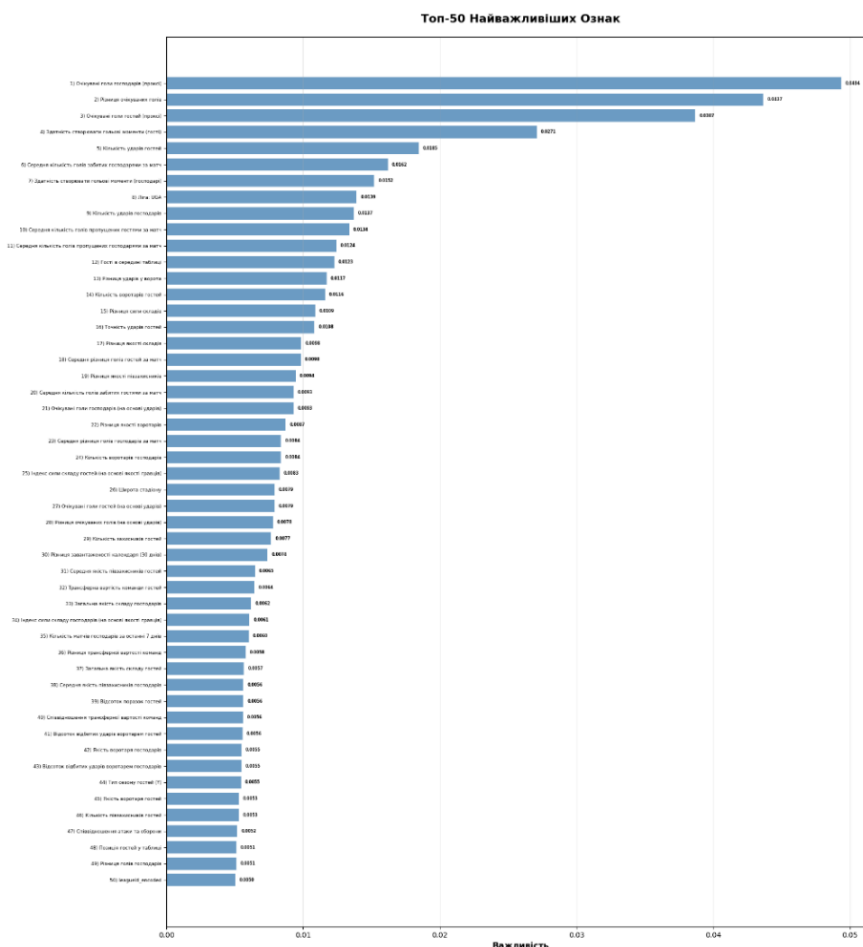


Рис. 2.8 – Розподіл важливості п’ятдесяти найважливіших ознак

Крім того, значний внесок мають ознаки, що характеризують поточну ігрову форму, зокрема кількість набраних очок у нещодавніх матчах, серії перемог або поразок. Такі характеристики дозволяють моделі враховувати динаміку виступів команд, а не лише їхні сезонні середні значення. Водночас допоміжні ознаки, наприклад статистика очних зустрічей, календарне навантаження, дні відпочинку мають меншу важливість, проте в сукупності уточнюють прогноз і підвищують його стабільність. Такий розподіл значущості ознак показує, що модель повинна спиратись як на основні показники сили команд, так і на додаткові характеристики матчу, а їхнє поєднання дозволить отримати більш точний прогноз.

Отже, у результаті проведеного відбору було сформовано оптимальний набір зі ста п’ятдесяти найбільш інформативних ознак, що забезпечує збалансоване співвідношення між точністю прогнозування та

узагальнювальною здатністю моделей. Надалі цей набір ознак буде використано для побудови, навчання та порівняльного аналізу моделей прогнозування.

2.3 Реалізація моделей прогнозування футбольних матчів та фінальної турнірної таблиці

Після завершення етапів збору, очищення та підготовки даних, а також формування та відбору інформативних ознак, наступним кроком є реалізація процесу прогнозування результатів футбольних матчів. На цьому етапі сформований датасет використовується для побудови, навчання та оцінювання моделей машинного навчання,

Прогнозування футбольних матчів є задачею навчання з учителем. На вхід модель отримує дані про команди та матч, а на виході передбачає результат гри. Передбачати можна різні речі, зокрема кількість голів кожної команди, переможця матчу, чи заб'ють обидві команди хоча б по голу, чи буде загальна кількість голів вищою за певний поріг. У зв'язку з цим у роботі розглядається кілька підходів до моделювання.

Першим кроком виконується впорядкування всіх матчів за датою проведення. Це важливо, оскільки футбольні події мають чітку часову послідовність, а використання інформації з майбутніх матчів під час навчання моделі призводить до витоку даних та некоректної оцінки якості прогнозування.

Після сортування датасет розбивається на три неперетинні підвибірки: навчальну, валідаційну та тестову у співвідношенні 70 %, 15 % та 15 %. Навчальна вибірка використовується безпосередньо для оптимізації параметрів моделей, валідаційна – для вибору найбільш ефективного варіанту моделі та контролю перенавчання, а тестова – для фінальної оцінки якості прогнозування.

Значення числових ознак можуть суттєво відрізнятися за масштабом. Наприклад, одні показники можуть набувати значень у межах кількох

одиниць, тоді як інші – десятків або сотень. Така різниця у масштабах може ускладнювати навчання моделей, оскільки ознаки з великими числовими значеннями починають домінувати над іншими. З цієї причини перед навчанням виконується нормалізація ознак. Перетворення виконується за формулою (2.17) [23]:

$$x_{norm} = \frac{x - \mu}{\sigma}, \quad (2.17)$$

де x – початкове значення ознаки;

μ – середнє значення цієї ознаки;

σ – стандартне відхилення ознаки.

Після такого перетворення значення різних ознак опиняються в одному масштабі. Це означає, що жодна ознака не має штучної переваги лише через великі числові значення. При цьому нормалізація не обмежує значення фіксованим інтервалом, а лише спрощує для моделі врахування впливу кожної ознаки під час навчання.

Першою реалізованою моделлю для прогнозування футбольних матчів є гібридна пуассонівська модель. Підхід базується на застосуванні регресійної моделі XGBoost для прогнозування кількості забитих голів кожною з команд. Далі отримані оцінки використовуються як параметри пуассонівського розподілу для обчислення ймовірностей можливих рахунків матчу. Отже, модель машинного навчання прогнозує, скільки голів можуть забити команди, а ймовірнісна модель перетворює ці прогнози на ймовірності конкретних результатів матчу.

Реалізація цього підходу виконана у вигляді окремого класу HybridPoissonModel, який об'єднує всі етапи прогнозування в єдиний послідовний процес. Під час ініціалізації класу передається навчена регресійна модель XGBoost, та сама модель, яка використовувалася на попередньому етапі для оцінювання важливості ознак, а також параметр max_goals, який

задає максимальну кількість голів, що враховується при побудові матриці ймовірних рахунків.

```
class HybridPoissonModel:
    def __init__(self, xgb_model, max_goals=8):
        self.xgb_model = xgb_model
        self.max_goals = max_goals
```

Першим етапом прогнозування є отримання очікуваної кількості голів для господарів і гостей. Для цього використовується метод `predict_lambdas`, який приймає вхідні ознаки матчу та передає їх у модель XGBoost

```
def predict_lambdas(self, X):
    if isinstance(X, pd.DataFrame):
        X = X.values
    if X.ndim == 1:
        X = X.reshape(1, -1)
    lambdas = self.xgb_model.predict(X)
    return lambdas
```

Після отримання параметрів інтенсивності будується матриця ймовірностей рахунків. Ця матриця має фіксований розмір `max_goals × max_goals`, де кожна комірка відповідає конкретному рахунку матчу. Значення в комірках обчислюються як добуток пуассонівських ймовірностей для господарів і гостей. Оскільки враховується лише скінченний діапазон рахунків, матриця додатково нормалізується, щоб сума всіх ймовірностей дорівнювала одиниці.

```
def generate_score_matrix(self, lambda_home, lambda_away):
    matrix = np.zeros((self.max_goals, self.max_goals))
    for h in range(self.max_goals):
        for a in range(self.max_goals):
            matrix[h, a] = poisson.pmf(h, lambda_home) * poisson.pmf(a,
lambda_away)

    return matrix / matrix.sum()
```

Отримана матриця рахунків використовується як основа для обчислення ймовірностей різних матчевих подій. Ймовірність перемоги господарів визначається як сума всіх комірок, у яких кількість голів господарів більша за кількість голів гостей. Ймовірність нічиєї відповідає сумі елементів головної діагоналі матриці, а ймовірність перемоги гостей – сумі комірок, де гості забили більше голів.

Подія «обидві команди заб'ють» обчислюється як доповнення до ймовірності того, що хоча б одна з команд не заб'є жодного м'яча. Для цього використовується інформація з першого рядка та першого стовпця матриці рахунків. Аналогічно обчислюються ймовірності перевищення різних порогів тоталу шляхом сумування комірок, у яких сумарна кількість голів перевищує задане значення.

```
def matrix_to_predictions(self, matrix):
    prob_home = np.sum(np.tril(matrix, -1))
    prob_draw = np.trace(matrix)
    prob_away = np.sum(np.triu(matrix, 1))

    prob_no_btts = matrix[0, :].sum() + matrix[:, 0].sum() - matrix[0, 0]
    prob_btts = 1 - prob_no_btts

    def calc_over(threshold):
        return sum(matrix[h, a] for h in range(self.max_goals)
                    for a in range(self.max_goals) if h + a > threshold)

    return {
        'outcome': np.array([prob_home, prob_draw, prob_away]),
        'btts': prob_btts,
        'over15': calc_over(1.5),
        'over25': calc_over(2.5),
        'over35': calc_over(3.5),
        'score_matrix': matrix
    }
```

У підсумку гібридна пуассонівська модель забезпечує повний процес прогнозування футбольних матчів на основі передматчевих ознак. Оцінки очікуваної кількості голів, отримані за допомогою моделі машинного навчання, використовуються для побудови ймовірностей результату матчу та інших цільових змінних таких як те, що обидві команди заб'ють, або загальна кількість голів буде більше 2.5 тощо.

Другою реалізованою моделлю прогнозування стала багатовихідна нейронна мережа. У цьому підході прогнозування футбольного матчу розглядається як задача одночасного передбачення кількох пов'язаних цільових змінних на основі одного набору передматчевих ознак. На вхід мережі подається нормалізований вектор характеристик матчу, після чого ці дані послідовно обробляються кількома повнозв'язними шарами. Нижче на рис. 2.9 схематично представлено архітектуру цієї нейронної мережі

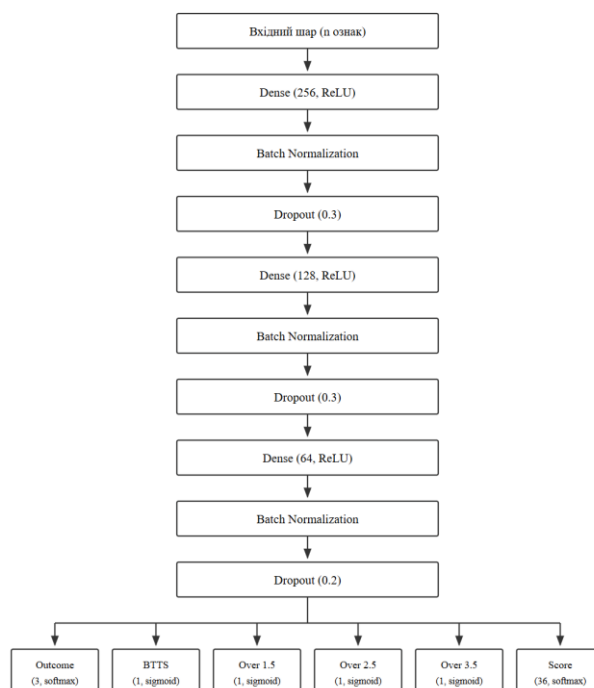


Рис. 2.9 – Архітектура багатовихідної нейронної мережі

Архітектура мережі складається з трьох прихованих шарів розмірності 256, 128 та 64 нейрони з нелінійною активацією ReLU. Кожен наступний шар має меншу кількість нейронів, що дозволяє поступово зменшувати розмірність даних і відокремлювати найбільш інформативні поєднання ознак. Після кожного прихованого шару застосовується пакетна нормалізація, яка стабілізує процес навчання та зменшує чутливість моделі до масштабу значень. Додатково використовується Dropout, який випадковим чином вимикає частину нейронів під час навчання і таким чином знижує ризик перенавчання.

Після проходження прихованих шарів мережа формує кілька вихідних значень. Частина з них відповідає за прогнозування результату матчу у вигляді ймовірностей перемоги господарів, нічиєї або перемоги гостей. Інші виходи використовуються для оцінювання ймовірностей подій, пов'язаних з кількістю забитих м'ячів, зокрема факту обміну голами між командами та перевищення загальної кількості голів певного значення.

Реалізація архітектури наведена нижче. Вона починається з вхідного шару, далі використовуються кілька повнозв'язних шарів з нормалізацією та

регуляризацією, після чого формуються окремі вихідні значення відповідного типу.

```
def build_nn(input_dim):
    inputs = layers.Input(shape=(input_dim,))

    x = layers.Dense(256, activation='relu')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.3)(x)

    x = layers.Dense(128, activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.3)(x)

    x = layers.Dense(64, activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.2)(x)

    outcome = layers.Dense(3, activation='softmax')(x)
    btts = layers.Dense(1, activation='sigmoid')(x)
    over15 = layers.Dense(1, activation='sigmoid')(x)
    over25 = layers.Dense(1, activation='sigmoid')(x)
    over35 = layers.Dense(1, activation='sigmoid')(x)
    score = layers.Dense(36, activation='softmax')(x)

    return keras.Model(
        inputs=inputs,
        outputs=[outcome, btts, over15, over25, over35, score]
    )
```

Третьою реалізованою моделлю прогнозування футбольних матчів є ансамблевий підхід на основі алгоритму випадкового лісу. На відміну від пуассонівської моделі та нейронної мережі, цей метод формує прогноз шляхом поєднання великої кількості дерев рішень, кожне з яких навчається незалежно на випадковій підмножині даних та ознак. Підсумковий прогноз визначається шляхом усереднення результатів окремих дерев, що дозволяє зменшити вплив шуму в даних і підвищити стабільність моделі.

У межах цієї роботи модель випадкового лісу навчається на повному наборі вхідних ознак, який містить велику кількість різномірних характеристик. До них входять турнірні показники команд, метрики поточної форми, агреговані показники складу та інші ознаки, що відображають різницю між суперниками. Випадковий ліс добре підходить у цій ситуації, оскільки кожне дерево під час навчання використовує лише випадкову підмножину характеристик, що знижує залежність моделі від окремих змінних і дозволяє враховувати складні нелінійні взаємозв'язки між ними.

Для практичної реалізації алгоритму випадкового лісу використовується клас `RandomForestClassifier` з бібліотеки `scikit-learn`. Усі моделі мають однакову конфігурацію гіперпараметрів, які представлені нижче, що забезпечує узгодженість результатів для різних цільових змінних.

```
rf_params = {  
    'n_estimators': 200,  
    'max_depth': 15,  
    'min_samples_split': 2,  
    'min_samples_leaf': 1,  
    'max_features': 'sqrt',  
    'bootstrap': True,  
    'random_state': 42,  
    'n_jobs': -1,  
    'verbose': 0  
}
```

Обрані параметри випадкового лісу визначають кількість дерев у моделі та обмежують складність кожного з них. Використання великої кількості дерев дозволяє зменшити вплив випадкових помилок окремих рішень, а обмеження глибини дерев запобігає надмірному пристосуванню до навчальних даних. Для побудови кожного дерева застосовується випадковий відбір частини ознак і навчальних прикладів, що підвищує різноманітність рішень в ансамблі. Такий підхід забезпечує стабільну роботу моделі та кращу здатність до узагальнення при прогнозуванні нових матчів.

Оскільки базова реалізація моделі випадкового лісу орієнтована на розв'язання однієї задачі класифікації, для кожної матчевої події навчається окрема модель. Зокрема, незалежні моделі використовуються для прогнозування результату матчу, події обміну голами тощо:

```
rf_outcome = RandomForestClassifier(**rf_params)  
rf_outcome.fit(X_train, y_train_outcome)  
  
rf_btts = RandomForestClassifier(**rf_params)  
rf_btts.fit(X_train, y_train_btts)  
  
rf_over25 = RandomForestClassifier(**rf_params)  
rf_over25.fit(X_train, y_train_over25)
```

Слід зазначити, що перед навчанням моделі, всі цільові змінні для кожного матчу були сформовані у навчальній вибірці на основі фактичних результатів матчів. Для результату матчу використовувалося дискретне

кодування трьох класів, що відповідають перемозі господарів, нічий та перемозі гостей, а для бінарних подій – відповідні бінарні мітки.

Важливою перевагою моделі випадкового лісу є те, що вона безпосередньо формує ймовірнісні прогнози. Кожне дерево в ансамблі повертає власну оцінку класів, а фінальна ймовірність визначається як середнє значення по всіх деревах. Отримання ймовірностей реалізується за допомогою стандартного методу `predict_proba`, який повертає розподіл імовірностей для кожного класу.

```
rf_outcome_probs = rf_outcome.predict_proba(X_val)
rf_btts_probs = rf_btts.predict_proba(X_val)[: , 1]
rf_over25_probs = rf_over25.predict_proba(X_val)[: , 1]
```

Так само, як і у випадку з моделлю XGBoost, алгоритм випадкового лісу дозволяє оцінити важливість вхідних ознак, однак робить це на основі власного внутрішнього механізму. У моделі випадкового лісу важливість ознак визначається на основі того, наскільки активно кожна з них використовується під час побудови рішень. На рисунку наведено перелік ознак, які модель вважає найбільш значущими для формування прогнозу. На рис. 2.10 наведено перелік найбільш важливих ознак за оцінкою моделі випадкового лісу.

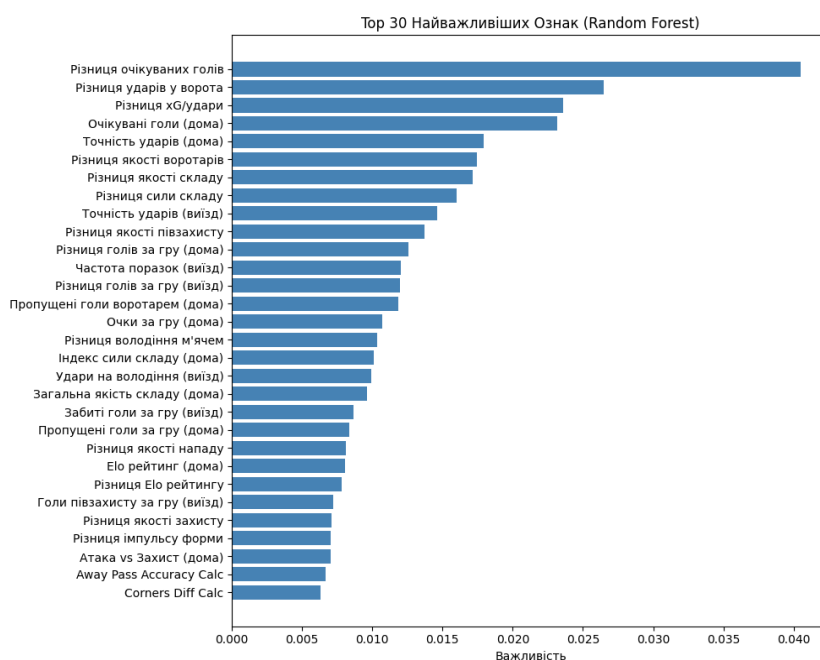


Рис. 2.10 – Перелік найважливіших ознак для моделі випадкового лісу

Наступним кроком є безпосереднє порівняння розглянутих моделей на валідаційній вибірці з метою визначення моделі, яка забезпечує найкращу якість прогнозів. Для коректного порівняння моделей буде використано набір метрик, орієнтованих не лише на правильність класифікації, а передусім на якість прогнозованих ймовірностей.

Базовою метрикою для такого аналізу є квадратична ймовірнісна похибка, відома як Brier Score. Вона вимірює середньоквадратичне відхилення між прогнозованими ймовірностями та фактичним результатом матчу і визначається наступною формулою (2.18) [24]:

$$BS = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K (p_{ik} - y_{ik})^2, \quad (2.18)$$

де N – кількість матчів;

K – кількість можливих результатів;

p_{ik} – прогнозована моделлю ймовірність результату k для матчу i ;

y_{ik} – фактичний індикатор результату.

На практиці це означає, що модель отримує штраф за кожне відхилення між тим, яку ймовірність вона призначила події, і тим, чи відбулася ця подія насправді. Якщо модель систематично завищує або занижує ймовірності окремих результатів, значення Brier Score зростає. Таким чином, ця метрика одночасно враховує як точність прогнозу, так і його калібрування.

Для бінарної події (наприклад, «обидві команди заб'ють») використовується спрощений варіант цієї ж метрики (2.19):

$$BS = \frac{1}{N} \sum_{i=1}^N (p_i - y_i)^2, \quad (2.19)$$

де p_i – прогнозована ймовірність події;

y_i – фактичний результат.

Додатково для оцінювання результату матчу застосовано Ranked Probability Score – рангову ймовірнісну похибку. Ця метрика є узагальненням Brier Score для впорядкованих класів і враховує не лише сам факт помилки, а й те, наскільки далеко прогноз знаходиться від реального результату.

Формально RPS для одного матчу обчислюється як (2.20) [25]:

$$RPS = \frac{1}{K-1} \sum_{k=1}^{K-1} (\sum_{j=1}^k p_j - \sum_{j=1}^k y_j)^2, \quad (2.20)$$

де p_j – прогнозовані ймовірності класів;

y_j – фактичний індикатор класу.

На практиці це означає, що помилка між, наприклад, перемогою господарів і нічиєю вважається менш серйозною, ніж помилка між перемогою господарів і перемогою гостей. Саме ця властивість робить RPS особливо корисною для футбольних результатів, де класи мають природний порядок. Як і у випадку з Brier Score, менше значення RPS відповідає кращій якості прогнозу. Результати порівняння якості прогнозування моделей наведено на рис. 2.11.

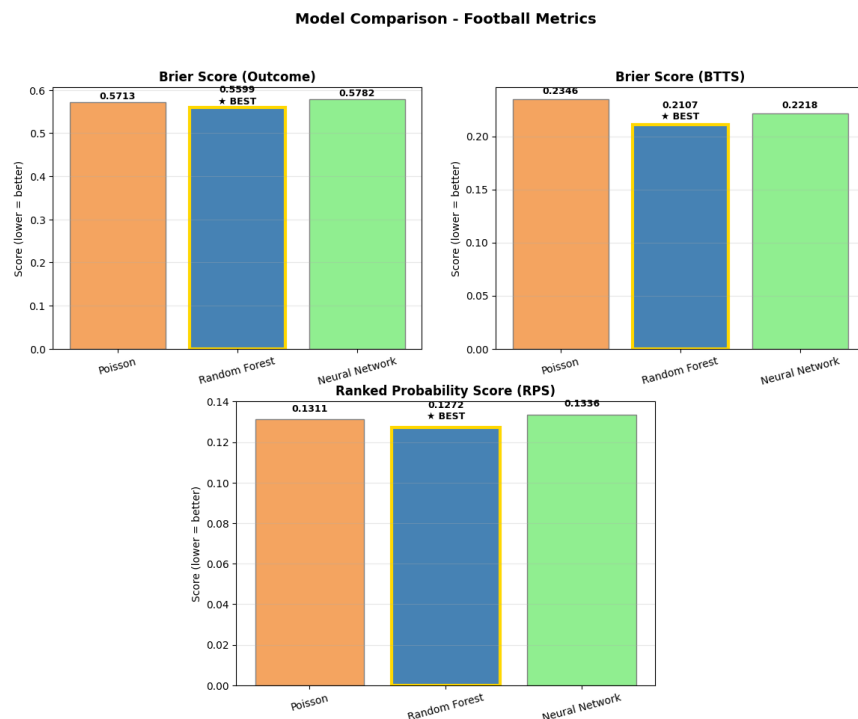


Рис. 2.11 – Результати порівняння якості прогнозування моделей

Аналіз значень Brier Score для результату матчу показав, що модель випадкового лісу формує більш точні ймовірнісні оцінки порівняно з нейронною мережею та пуассонівською моделлю. Аналогічна тенденція спостерігається і для події обидві команди заб'ють, де випадковий ліс демонструє нижче значення квадратичної похибки. Значення Ranked Probability Score також свідчать про перевагу моделі випадкового лісу. Це означає, що її ймовірнісні прогнози не лише ближчі до фактичних результатів, але й краще враховують порядок можливих наслідків матчу.

У підсумку можна зробити висновок, що серед розглянутих підходів модель випадкового лісу демонструє найкращу загальну якість ймовірнісного прогнозування. З практичної точки зору це означає, що саме модель випадкового лісу є найбільш доцільною для реалізації основної логіки прогнозування матчевих подій, таких як результат матчу або обмін голами. Водночас для прогнозування точної кількості забитих голів доцільніше використовувати пуассонівську модель, яка краще відповідає природі цієї задачі.

Окрім прогнозування результатів окремих матчів, важливою задачею спортивної аналітики є формування ймовірнісного прогнозу фінальної турнірної таблиці. На відміну від короткострокових прогнозів поєдинків, ця задача потребує врахування довгострокової динаміки сезону, послідовності результатів та взаємного впливу команд протягом усього чемпіонату. Для вирішення цієї задачі було розроблено модель на основі рекурентної нейронної мережі типу LSTM, яка здатна ефективно обробляти послідовності матчів та враховувати часові залежності між ними.

Задача прогнозування фінальної турнірної таблиці формулюється наступним чином: для кожної команди ліги необхідно визначити розподіл ймовірностей зайняття кожної позиції в підсумковій таблиці. Результатом роботи моделі є матриця розміром $N \times N$, де N – кількість команд у лізі. Кожен рядок матриці відповідає одній команді, а кожен стовець – позиції в таблиці. Значення комірки відображає ймовірність того, що команда займе певну

позицію. При цьому сума ймовірностей у кожному рядку та кожному стовпці має дорівнювати 100%, що забезпечує коректність ймовірнісного розподілу.

Ключовим елементом підготовки даних для моделі є snapshot концепція зрізу стану ліги на певний момент сезону. Кожен snapshot містить інформацію про всі команди ліги, зокрема, послідовності матчів, контекстні вектори та фінальні позиції в турнірній таблиці. Зрізи створюються на різних етапах сезону, що дозволяє моделі навчатися прогнозувати фінальну таблицю з різних точок сезону – як на його початку, так і ближче до завершення.

Архітектура розробленої моделі складається з двох основних компонентів, а саме моделі для обробки даних однієї команди та моделі-обгортки, що застосовує першу модель до всіх команд ліги.

Нижче наведено реалізацію архітектури моделі для однієї команди:

```
def build_lstm_model(n_teams, seq_length, n_features, context_dim):
    seq_input = Input(shape=(seq_length, n_features),
name='sequence_input')
    ctx_input = Input(shape=(context_dim,), name='context_input')

    lstm = LSTM(64, return_sequences=False, dropout=0.2)(seq_input)
    lstm = BatchNormalization()(lstm)
    lstm = Dropout(0.3)(lstm)

    combined = Concatenate()([lstm, ctx_input])
    dense = Dense(64, activation='relu')(combined)
    dense = Dropout(0.2)(dense)
    output = Dense(n_teams, activation='softmax')(dense)

    return Model(inputs=[seq_input, ctx_input], outputs=output)
```

Модель для однієї команди отримує два входи. Перший вхід seq_input це двовимірний масив з результатами матчів команди у турнірі. Наприклад, якщо команда зіграла 10 матчів і кожен матч описаний 15 ознаками, то масив має розмір 10 на 15. Ознаки включають забиті та пропущені голи, удари по воротах, очікувані голи тощо. Другий вхід ctx_input це одновимірний масив з агрегрованою інформацією про становище команди в турнірі на момент прогнозування, а саме позицію в таблиці, кількість очок, різницю забитих й пропущених голів тощо.

Рекурентний шар типу LSTM виконує послідовну обробку матчів команди, аналізуючи їх у хронологічному порядку. На кожному кроці він

оновлює свій внутрішній стан, який зберігає інформацію про попередні ігри, що дозволяє враховувати не окремі матчі, а їхню динаміку у часі. Завдяки внутрішнім механізмам керування пам'яттю, LSTM здатний зберігати важливі довгострокові закономірності, наприклад зміни форми команди або стабільність результатів протягом сезону. Фінальний прихований стан шару є узагальненим представленням усієї історії матчів команди та використовується для подальшого формування прогнозу

Після обробки історії матчів рекурентним шаром формується вектор, який узагальнює інформацію про виступи команди. Цей вектор спочатку нормалізується, щоб значення ознак мали стабільний масштаб, а потім проходить через шар регуляризації, який випадково вимикає частину нейронів. Такий підхід зменшує залежність моделі від окремих ознак і допомагає уникнути перенавчання. Після додаткової регуляризації результат передається на фінальний шар з функцією активації Softmax, який перетворює вихідні значення на ймовірності можливих підсумкових позицій команди.

Оскільки вихідна матриця ймовірностей, сформована моделлю, не гарантує виконання обмеження, що сума ймовірностей на одну позицію буде дорівнювати одиниці, додатково застосовується спеціальний алгоритм корекції розподілу. Наведена функція виконує корекцію матриці ймовірностей з метою узгодження розподілу між командами та турнірними позиціями. Цей алгоритм ітеративно масштабує значення матриці, по чергово нормалізуючи рядки та стовпці, доки суми значень у них не наближаються до одиниці.

```
def normalization(matrix, num_iterations=100, epsilon=1e-8):  
    matrix = matrix.copy()  
    for _ in range(num_iterations):  
        matrix = matrix / (matrix.sum(axis=1, keepdims=True) + epsilon)  
        matrix = matrix / (matrix.sum(axis=0, keepdims=True) + epsilon)  
  
    return matrix
```

Після завершення навчання моделі також було виконано її оцінювання за допомогою двох ймовірнісних метрик – Brier Score та Ranked Probability Score, які дозволяють оцінити не лише правильність прогнозу, а й якість розподілу ймовірностей (рис. 2.12).

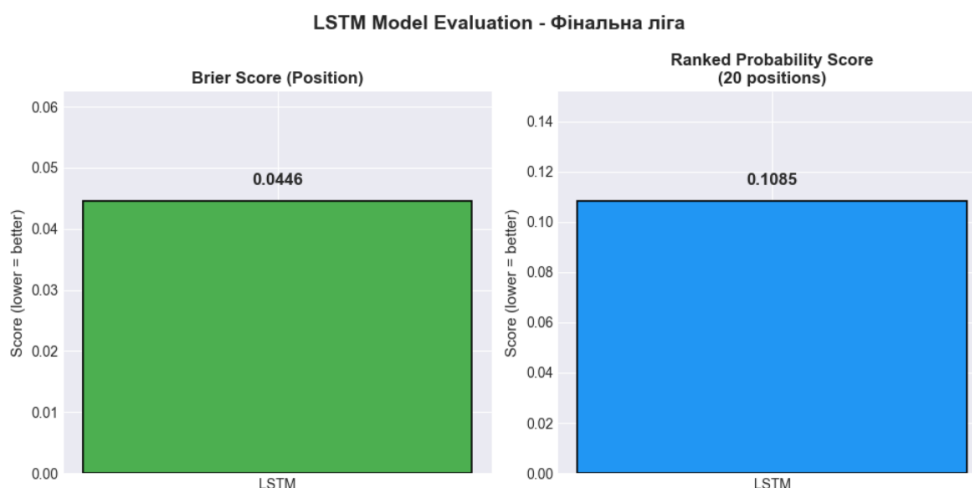


Рис. 2.12 – Результати оцінки якості прогнозування LSTM моделі для турнірної таблиці

Отримане значення Brier Score ≈ 0.045 на валідаційній вибірці свідчить про невелику середньоквадратичну похибку між прогнозованими й фактичними позиціями команд, що означає достатньо точну калібровку ймовірностей. Значення Ranked Probability Score ≈ 0.109 , розраховане для, в середньому, двадцяти можливих позицій, показує, що модель коректно враховує порядок позицій у турнірній таблиці та надає вищі ймовірності позиціям, близьким до фактичного результату. У сукупності ці показники підтверджують, що модель формує узгоджений і стабільний ймовірнісний прогноз фінального розташування команд.

Висновки до розділу 2

У другому розділі було розроблено комплексні моделі прогнозування результатів футбольних матчів та фінальної турнірної таблиці.

На етапі підготовки даних було сформовано датасет на основі відкритих даних платформи Kaggle, який містить історичні матчі з детальною статистикою команд та гравців. Виконано очищення даних та обробку пропущених значень. Також розроблено систему конструювання ознак, що включає турнірні показники, метрики поточної форми команд, агреговані характеристики складу та показники очікуваних голів.

Для прогнозування результатів окремих матчів реалізовано та порівняно три моделі машинного навчання, а саме гібридну пуассонівську модель на основі XGBoost, багатовихідну нейронну мережу та ансамблеву модель випадкового лісу. Порівняння моделей за метриками Brier Score та Ranked Probability Score показало, що модель випадкового лісу демонструє найкращу якість ймовірнісного прогнозування результату матчу та події обміну голами, тоді як пуассонівська модель є оптимальною для прогнозування точного рахунку.

Для прогнозування фінальної турнірної таблиці була розроблена модель на основі рекурентної нейронної мережі LSTM. Оцінювання на валідаційній вибірці показало, що отримані помилки є невеликими, а прогнозовані ймовірності добре узгоджуються з фактичними результатами.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІТИКИ ФУТБОЛЬНИХ МАТЧІВ

3.1 Розробка програмного рішення для практичного застосування моделей прогнозування

Система прогнозування результатів футбольних матчів спроектована за принципами багатoshарової архітектури з розділенням відповідальності між основними рівнями, такими як рівень представлення, рівень бізнес-логіки, рівень моделей машинного навчання, а також рівень даних. Такий підхід дозволяє ізолювати отримання та агрегацію даних від обчислювально-затратної частини прогнозування, а також спрощує тестування і подальшу заміну окремих модулів без необхідності внесення змін до всієї системи.

На рис. 3.1 наведено узагальнену діаграму компонентів та їхньої взаємодії.

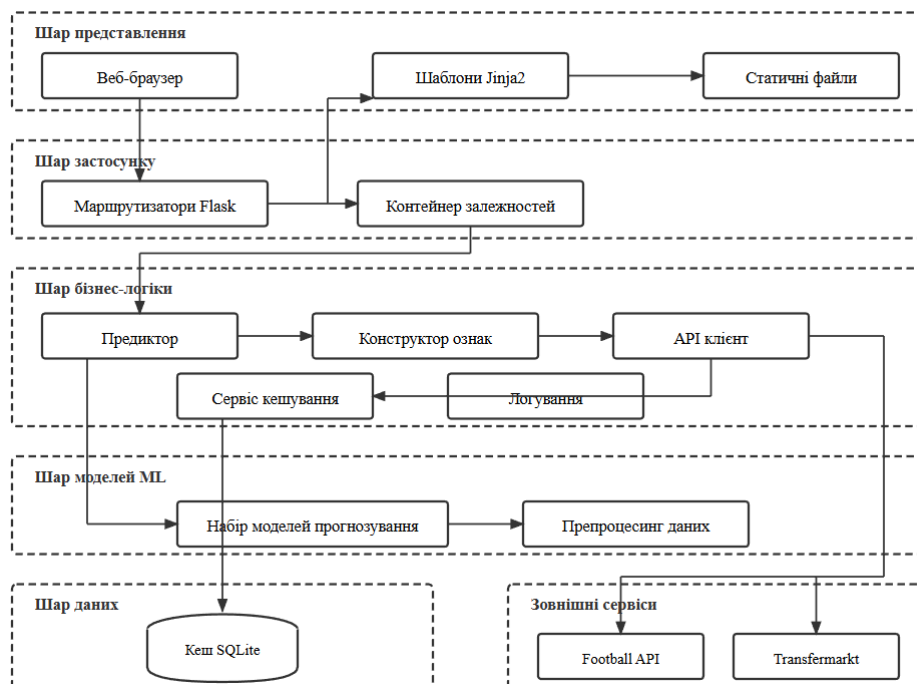


Рис. 3.1 – Узагальнена діаграма компонентів та їхньої взаємодії

Шар представлення забезпечує взаємодію з користувачем через веббраузер, формуючи HTML-сторінки за допомогою шаблонізатора Jinja2. Шар застосунку реалізований на базі Flask із використанням патерну Blueprint

для модульної організації маршрутів та контейнера залежностей для централізованого управління сервісами.

Шар бізнес-логіки містить основні компоненти системи: сервіс прогнозування, що координує роботу моделей, конструктор ознак для формування вхідних даних, API-клієнт для отримання даних із зовнішніх джерел та сервіс кешування для зменшення навантаження на зовнішній API. Шар моделей інкапсулює навчені моделі машинного навчання, а шар даних представлений локальною базою SQLite для кешування. Такі зовнішні сервіси як Football API та Transfermarkt слугують джерелами актуальної інформації про футбольні матчі та команди.

Контейнер залежностей є центральним елементом архітектури, що відповідає за створення та управління життєвим циклом усіх сервісів системи. Реалізація базується на шаблоні Declarative Container, де залежності описуються декларативно через провайдери [26]. Основними типами провайдерів є Singleton для сервісів, що мають існувати в єдиному екземплярі, зокрема це сервіси для логування, кешування, запитів до зовнішніх API [27].

Ієрархія залежностей побудована від простого до складного: спочатку створюються базові сервіси, які потім передаються у більш складні компоненти. Нижче наведено фрагмент реалізації контейнера:

```
class Container(containers.DeclarativeContainer):
    config = providers.Configuration()
    logger = providers.Singleton(Logger)

    cache_service = providers.Singleton(
        CacheService, logger=logger)

    api_client = providers.Singleton(
        ApiFootballClient,
        api_key=config.api_key,
        logger=logger,
        cache_service=cache_service)

    predictor = providers.Singleton(
        Predictor,
        model_bundle=model_bundle,
        feature_builder=feature_builder)
```

Клієнт API відповідає за взаємодію із зовнішнім сервісом Football API, який є основним джерелом статистичних даних про матчі та команди. Основні

функції клієнта включають формування HTTP-запитів та централізовану обробку, а також взаємодію з кешем. Нижче наведено базовий метод виконання запиту:

```
def _make_request(self, url: str, params: dict) -> dict:
    headers = {"x-apisports-key": self.api_key}
    try:
        resp = requests.get(url, params=params,
                             headers=headers, timeout=15)
        resp.raise_for_status()
        return resp.json()
    except requests.RequestException as e:
        self.logger.error(f"API request failed: {e}")
        return {}
```

Кешування використовується для зменшення кількості звернень до зовнішнього API, стабілізації часу відповіді вебінтерфейсу та повторного використання агрегованих даних. Кеш реалізовано за допомогою бази даних SQLite, зокрема було створено таблиці `fixture_cache` для зберігання даних про матчі та їхні прогнози й `data_cache` для агрегованих даних таких, як історія особистих зустрічей, турнірні таблиці тощо за унікальним ключем. На рис. 3.2 наведено структуру бази даних застосунку.

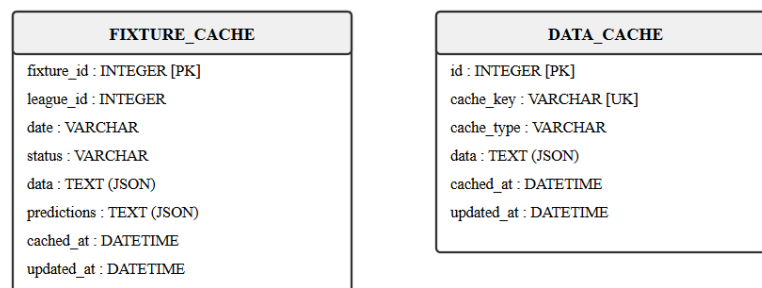


Рис. 3.2 – Структура бази даних застосунку

Система використовує стратегію `cache-aside`, за якої при запиті даних спочатку перевіряється наявність актуального запису в кеші, і лише за його відсутності або застарілості виконується запит до зовнішнього API з подальшим збереженням результату [28]. Терміни життя кешу налаштовані по-різному для різних типів даних: турнірні таблиці оновлюються частіше, а історія особистих зустрічей – рідше.

```
def get_cached_data(self, cache_type: str, **kwargs):
    key = DataCache.create_key(cache_type, **kwargs)
    row = session.query(DataCache)
```

```

        .filter_by(cache_key=key).first()
    if not row:
        return None
    if self._is_expired(row.cached_at, hours=TTL):
        return None
    return json.loads(row.data)

```

Сервіс побудови ознак FeatureBuilder об'єднує дані з API через клієнта, обчислює необхідні характеристики та формує вектор ознак для подальшого використання моделями. Типовий набір ознак включає дані турнірної таблиці, такі як позиція, очки, різниця голів, статистику голів, форму команд за останні N матчів, історію особистих зустрічей, а також обчислювальні ознаки між показниками домашньої та гостьової команд. Нижче наведено неповний лістинг функції, що відповідає за побудову ознак.

```

def build_features(
    self,
    home_team_id: int,
    away_team_id: int,
    league_id: int,
    season: int,
    fixture_date: str,
) -> dict:
    features = {}

    features["rank_diff"] = home_standing["rank"] - away_standing["rank"]
    features["points_diff"] = home_standing["points"] -
    away_standing["points"]

    home_goals = self.api_client.get_team_goals(home_team_id, league_id,
    season)
    away_goals = self.api_client.get_team_goals(away_team_id, league_id,
    season)

    home_gf_pg = home_goals["gf"] / max(home_goals["played"], 1)
    away_gf_pg = away_goals["gf"] / max(away_goals["played"], 1)

    features["home_gf_per_game"] = home_gf_pg
    features["away_gf_per_game"] = away_gf_pg
    features["gf_per_game_diff"] = home_gf_pg - away_gf_pg

    h2h_matches = self.api_client.get_head_to_head(home_team_id,
    away_team_id)

    if h2h_matches:
        home_wins = sum(1 for m in h2h_matches if m["winner"] == "home")
        features["h2h_home_win_rate"] = home_wins / len(h2h_matches)
    else:
        features["h2h_home_win_rate"] = 0.0

    return features

```

Рівень моделей інкапсулює навчені моделі машинного навчання та забезпечує їхнє завантаження, нормалізацію вхідних даних та отримання прогнозів. Компонент `ModelBundle` є фасадом, що надає уніфікований інтерфейс доступу до набору моделей. Варто відмітити, що моделі завантажуються один раз при старті застосунку та зберігаються в пам'яті протягом усього часу роботи, що забезпечує швидке виконання прогнозів без накладних витрат на повторне завантаження.

```
class ModelBundle:
    def __init__(self, base_dir: str):
        self.scaler = joblib.load(
            os.path.join(base_dir, "scaler.pkl"))
        self.outcome_model = joblib.load(
            os.path.join(base_dir, "rf_outcome.pkl"))
        self.bttts_model = joblib.load(
            os.path.join(base_dir, "rf_bttts.pkl"))

    def predict(self, features: dict) -> dict:
        X = self._to_vector(features)
        Xs = self.scaler.transform([X])

        outcome = self.outcome_model.predict_proba(Xs)
        bttts = self.bttts_model.predict_proba(Xs)

        return {"outcome": outcome, "bttts": bttts}
```

Сервіс прогнозування `Predictor` виступає фасадом для вебрівня, координуючи взаємодію між компонентами системи [29]. Він приймає ідентифікатори команд, ліги, сезону та дату матчу, викликає побудову ознак через `FeatureBuilder`, передає сформований вектор ознак у `ModelBundle` для отримання прогнозів, та формує єдину структуру відповіді для контролера.

Послідовність виконання прогнозу, яку зображено на рис. 3.3, починається із запиту користувача через браузер, який надходить до Flask-контролера [30]. Контролер викликає метод `predict` сервісу прогнозування, передаючи необхідні параметри. Сервіс прогнозування делегує побудову ознак компоненту `FeatureBuilder`, який звертається до `ApiFootballClient` для отримання статистики. Клієнт перевіряє наявність даних у кеші та за потреби виконує запит до зовнішнього API. Сформований вектор ознак передається до `ModelBundle`, де відбувається безпосереднє прогнозування. Результати

агрегуються та повертаються користувачу у вигляді HTML-сторінки або JSON-відповіді.

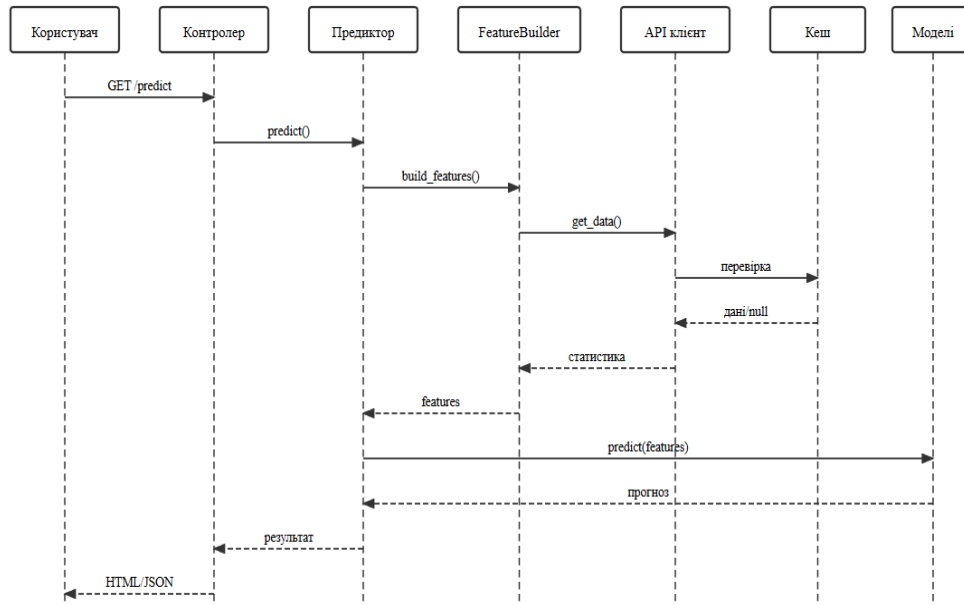


Рис. 3.3 – Послідовність виконання прогнозу

Отже, було реалізовано масштабоване програмне рішення для практичного застосування моделей прогнозування футбольних матчів та фінальних турнірних таблиць. Архітектура системи забезпечує можливість розширення, інтеграції нових моделей і джерел даних та підтверджує придатність розробленої системи до реального використання.

3.2 Опис користувацької взаємодії з розробленим програмним рішенням

Взаємодія користувача з програмним рішенням реалізована у вигляді вебсайту, який надає доступ до основних функцій системи через набір взаємопов'язаних сторінок. Робота з програмним рішенням починається з головної сторінки (рис. 3.4), яка слугує точкою входу до основних розділів системи.

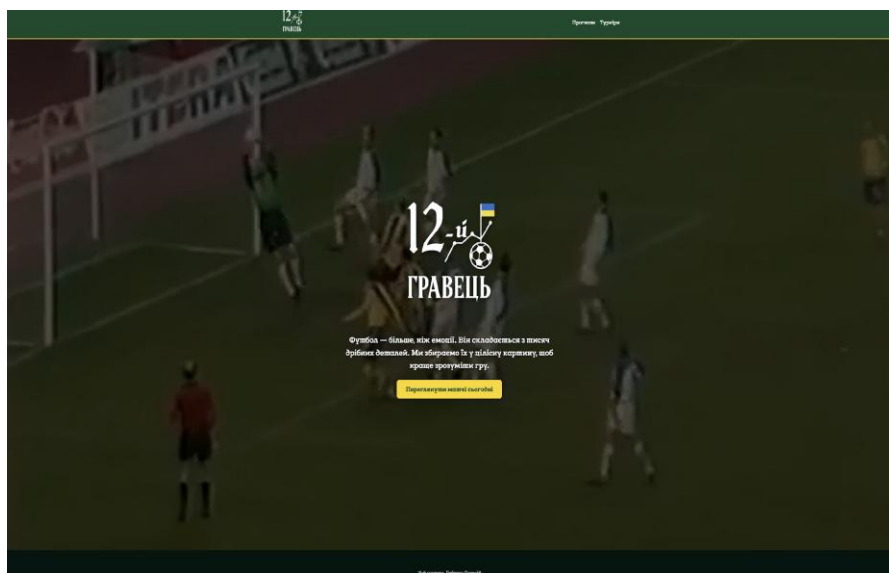


Рис. 3.4 – Головна сторінка вебсайту

Під час переходу на сторінку прогнозів матчів користувачу відображається перелік матчів, запланованих на обрану дату (рис. 3.5). У верхній частині сторінки розташовано елементи керування датою, зокрема кнопки для швидкого переходу між днями та селектор дати, що дозволяє змінювати день перегляду без переходу на інші сторінки. Поточна обрана дата відображається у заголовку сторінки.

Нижче список матчів згрупований за лігами. Кожна ліга відображається окремим блоком із назвою та логотипом, а також може бути розгорнута або згорнута користувачем для зручності навігації. Усередині кожної ліги матчі подані у вигляді окремих карток.

Картка матчу містить основну інформацію про зустріч, зокрема час початку, назви команд та їхні логотипи. Під назвами команд розміщено візуалізацію прогнозу результату матчу у вигляді горизонтальної кольорової смуги. Смуга поділена на три сегменти, які відповідають імовірності перемоги господарів, нічиєї та перемоги гостей. Довжина кожного сегмента відображає відповідну ймовірність, а кольори сегментів відповідають кольорам команд або нейтральному кольору для нічиєї.

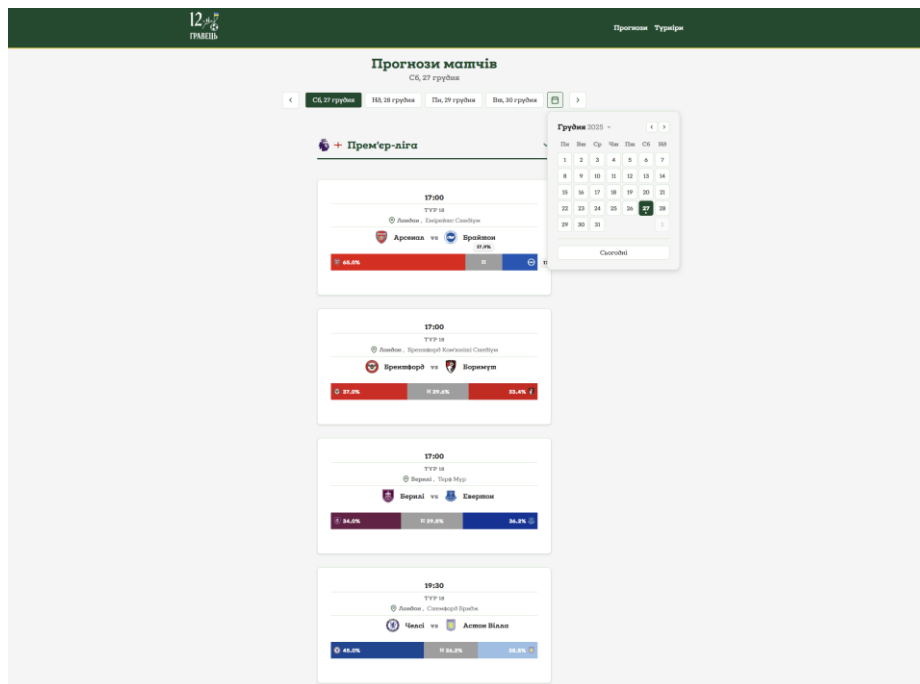


Рис. 3.5 – Сторінка прогнозів матчів

Для завершених матчів, які відображені на рис. 3.6, замість прогнозової смуги відображається фінальний рахунок, при цьому команда-переможець додатково виділяється кольором. Кожна картка матчу є інтерактивною: натискання на неї здійснює перехід до сторінки деталей відповідного матчу з розширеною інформацією та прогнозами.

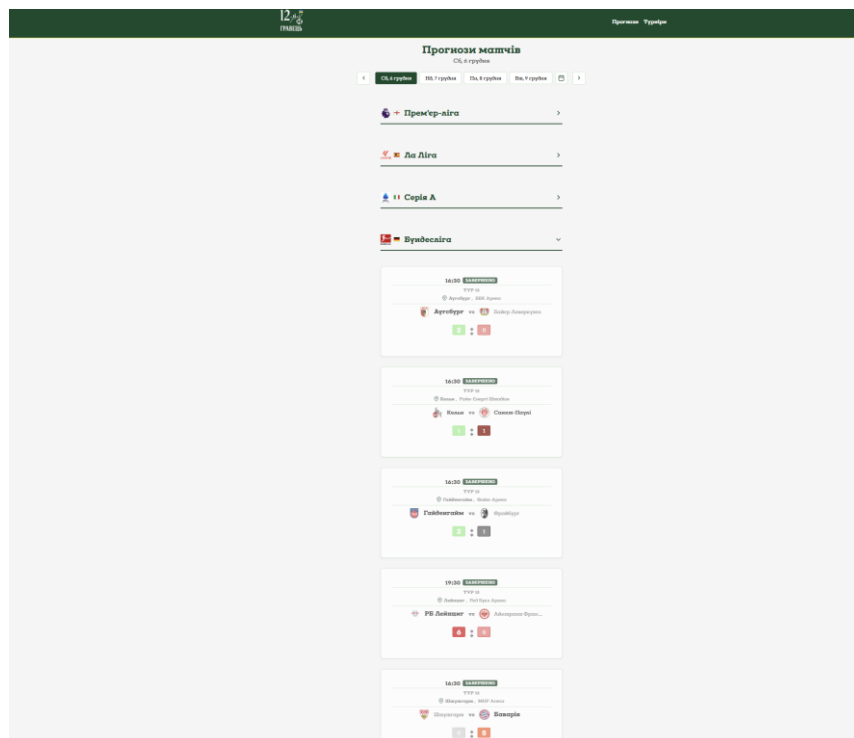


Рис. 3.6 – Сторінка завершених матчів

Для завершених матчів замість прогнозової смуги відображається фінальний рахунок, при цьому команда-переможець додатково виділяється кольором. Кожна картка матчу є інтерактивною, бо натискання на неї здійснює перехід до сторінки деталей відповідного матчу з розширеною інформацією та аналітикою.

Після вибору конкретного матчу зі списку користувач переходить на сторінку деталей матчу (рис. 3.7). Ця сторінка призначена для детального аналізу окремої зустрічі та містить як прогнозну, так і довідкову інформацію. У верхній частині сторінки відображається загальна інформація про матч, зокрема назва ліги, дата і час проведення, а також назви команд із їхніми логотипами.

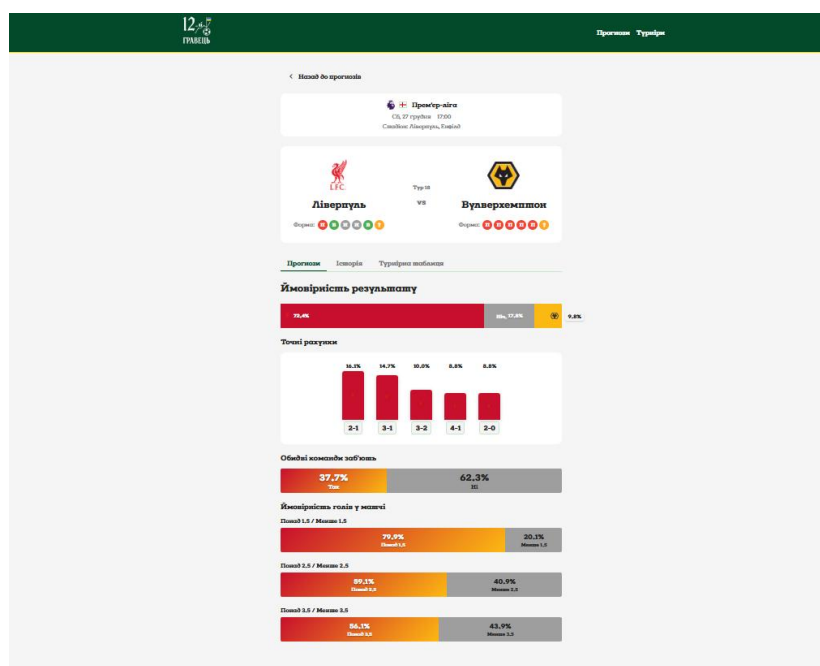


Рис. 3.7 – Сторінка прогнозу конкретного матчу

Основна частина сторінки організована у вигляді вкладок, що дозволяє структурувати інформацію за типами та спростити навігацію. За замовчуванням відкривається вкладка прогнозу, яка містить результати прогнозування для обраного матчу. У цьому розділі відображається ймовірність основних результатів матчу у вигляді горизонтальної смуги. Нижче подано блок з точними рахунками, який містить п'ять найбільш

ймовірні варіантів фінального рахунку з відповідними значеннями ймовірностей.

Окремо на вкладці прогнозу представлено оцінку ймовірності того, що обидві команди заб'ють, а також прогнози загальної кількості голів у матчі для порогових значень 1.5, 2.5 та 3.5. Уся інформація подається у вигляді наочних графічних елементів, що полегшує її сприйняття та порівняння. Наступна вкладка містить інформацію про попередні очні зустрічі команд за останні п'ять років, а вкладка з турнірною таблицею відображає поточне положення обох команд у таблиці відповідної ліги.

Перехід до сторінки окремої ліги можливий з навігаційного меню вебсайту. Сторінка ліги призначена для узагальненого перегляду інформації про конкретний турнір та аналізу матчів і команд у межах одного сезону (рис. 3.8).

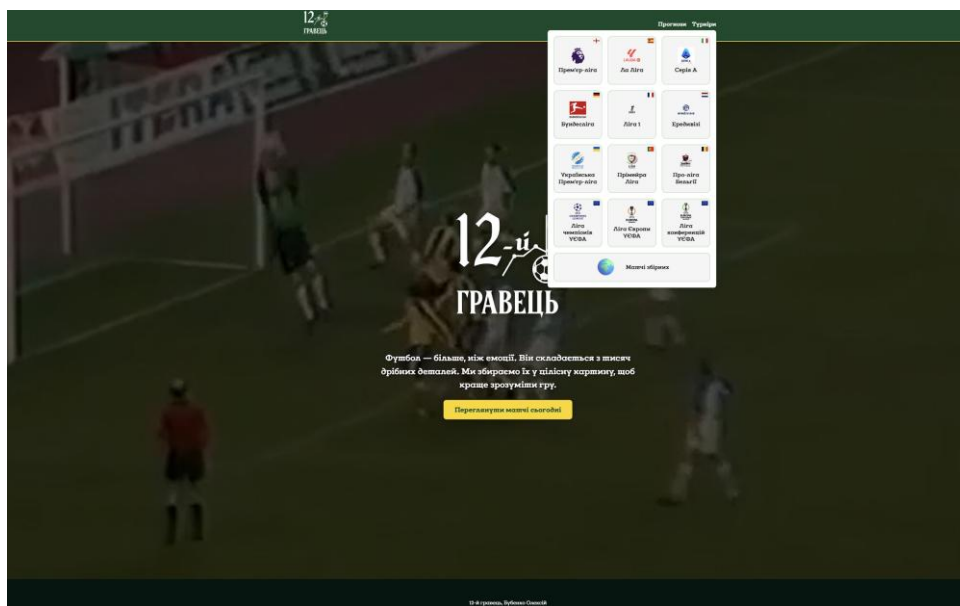


Рис. 3.8 – Перехід до сторінки окремої ліги через навігаційне меню

На рис.3.9 представлено сторінку ліги у вкладці прогнозів матчів, яка призначена для перегляду прогнозів на матчі обраного туру. У верхній частині сторінки відображається назва розділу, а також інформаційний блок ліги, що містить логотип турніру, його назву та прапор країни. Праворуч розташований селектор сезону, який дозволяє користувачу швидко перемикатися між різними сезонами чемпіонату.

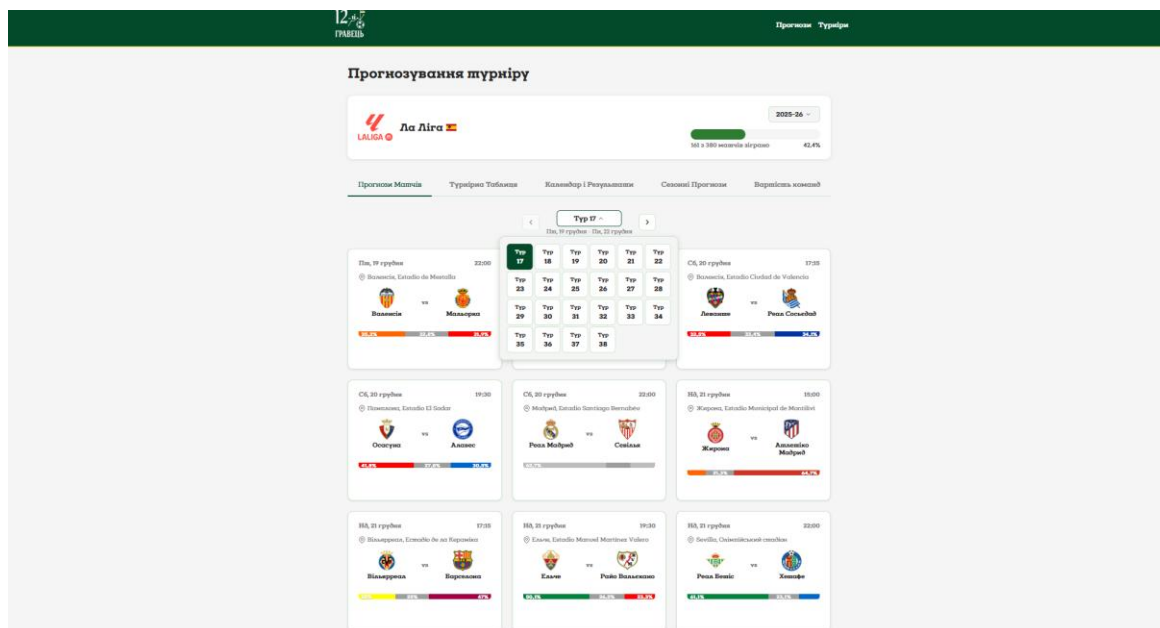


Рис. 3.9 – Сторінка ліги з картками матчів

Під інформаційним блоком розміщено індикатор прогресу сезону, який відображає кількість зіграних матчів від загальної кількості та відповідний відсоток. Це дозволяє оцінити поточну стадію турніру без переходу до додаткових розділів.

Нижче розташована панель вкладок, за допомогою якої користувач може перемикатися між основними розділами сторінки ліги: прогнозами матчів, турнірною таблицею, календарем і результатами, сезонними прогнозами та інформацією про вартість команд. Активна вкладка візуально виділяється.

У центральній частині сторінки розміщено блок навігації по турах. Користувач може змінювати тур за допомогою кнопок переходу або через селектор туру. Поруч відображається діапазон дат, у межах яких проходять матчі обраного туру.

Матчі обраного туру представлені у вигляді сітки карток. Кожна картка містить дату та час початку матчу, інформацію про місце проведення, назви команд та їхні логотипи. У нижній частині картки відображається прогноз результату у вигляді горизонтальної кольорової смуги, поділеної на сегменти.

Вкладка турнірної таблиці призначена для перегляду поточного розташування команд у межах обраної ліги (рис. 3.10). У таблиці відображено всі команди турніру з їхніми основними статистичними показниками, зокрема

кількістю зіграних матчів, перемог, нічий, поразок, різницею м'ячів та набраними очками. Команди впорядковані за місцем у таблиці відповідно до регламенту змагань.

Турнірна таблиця

Всі Вдома На виїзді Динаміка

#	Команда	Г	В	Н	П	Р	О	Останні 6
1	Барселона	17	14	1	2	29	43	●●●●●●●
2	Реал Мадрид	17	12	3	2	18	39	●●●●●●●
3	Вільярреал	15	11	2	2	18	35	●●●●●●●
4	Атлетіко Мадрид	17	10	4	3	14	34	●●●●●●●
5	Еспаньйол	16	9	3	4	4	30	●●●●●●●
6	Реал Бетіс	16	6	7	3	6	25	●●●●●●●
7	Атлетік Більбао	17	7	2	8	-7	23	●●●●●●●
8	Севілья Віго	16	5	7	4	1	22	●●●●●●●
9	Севілья	16	6	2	8	0	20	●●●●●●●
10	Хетафе	16	6	2	8	-5	20	●●●●●●●
11	Ельче	16	4	7	5	-1	19	●●●●●●●
12	Алавес	16	5	3	8	-3	18	●●●●●●●
13	Реал Вальєкано	16	4	6	6	-3	18	●●●●●●●
14	Малаґорка	16	4	5	7	-5	17	●●●●●●●
15	Реал Сосьєдад	16	4	4	8	-4	16	●●●●●●●
16	Осасуна	16	4	3	9	-6	15	●●●●●●●
17	Валенсія	16	3	6	7	-10	15	●●●●●●●
18	Жироа	16	3	6	7	-15	15	●●●●●●●
19	Огієда	16	2	4	10	-19	10	●●●●●●●
20	Леванте	15	2	3	10	-12	9	●●●●●●●

Легенда:
● Ліга чемпіонів ● Ліга Європи ● Ліга конференцій ● Най-оф. матч ● Вилет

Рис. 3.10. Турнірна таблиця ліги

Для зручності аналізу передбачені фільтри, які дозволяють переглядати статистику за всіма матчами, лише домашніми або лише виїзними іграми. При зміні фільтра таблиця автоматично оновлюється, а показники команд перераховуються відповідно до обраного режиму перегляду.

Окремою опцією є режим динаміки, зображений на рис. 3.11, який відображає зміну позицій команд протягом сезону у вигляді лінійного графіка. Кожна команда представлена окремою лінією, що дозволяє наочно простежити її рух у турнірній таблиці від туру до туру. Для полегшення ідентифікації використовуються кольори команд та їхні логотипи, а також інтерактивна легенда, яка дозволяє виділяти окремі команди.

Вкладка сезонного прогнозу призначена для оцінювання ймовірного підсумкового розташування команд у турнірній таблиці за результатами всього сезону. На відміну від прогнозів окремих матчів, цей розділ відображає довгостроковий сценарій розвитку турніру (рис. 3.13).

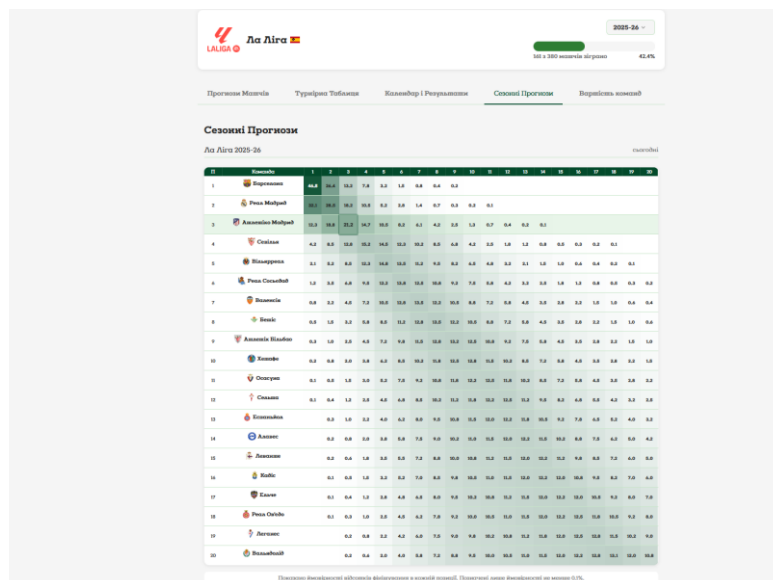


Рис. 3.13 – Сторінка сезонного прогнозу турніру

Основним елементом сторінки є таблиця, у якій для кожної команди показано ймовірності зайняття конкретних позицій у підсумковій турнірній таблиці. Стовпці відповідають місцям, а значення в клітинках відображають відсоткову ймовірність фінішу команди на відповідній позиції.

Для зручності сприйняття дані подані у вигляді теплової карти: чим вища ймовірність певної позиції, тим інтенсивніше зафарбована відповідна комірка. Це дозволяє швидко виявити фаворитів чемпіонату, команди середини таблиці та претендентів на зону вильоту.

У вкладці календаря і результатів, яку зображено на рис. 3.14, відображається повна інформація про проведені та заплановані матчі ліги. Матчі згруповані за турами, для кожного поєдинку вказані команди-учасники, час початку або фінальний рахунок у разі завершення матчу. Такий формат подання дозволяє швидко орієнтуватися в розкладі та результатах змагань.

Окрім актуальних даних, у системі доступні історичні результати. Користувач може переглядати матчі попередніх турів і сезонів, що дає змогу

відстежувати зміну позицій команд та загальну динаміку турніру без необхідності звернення до сторонніх джерел.

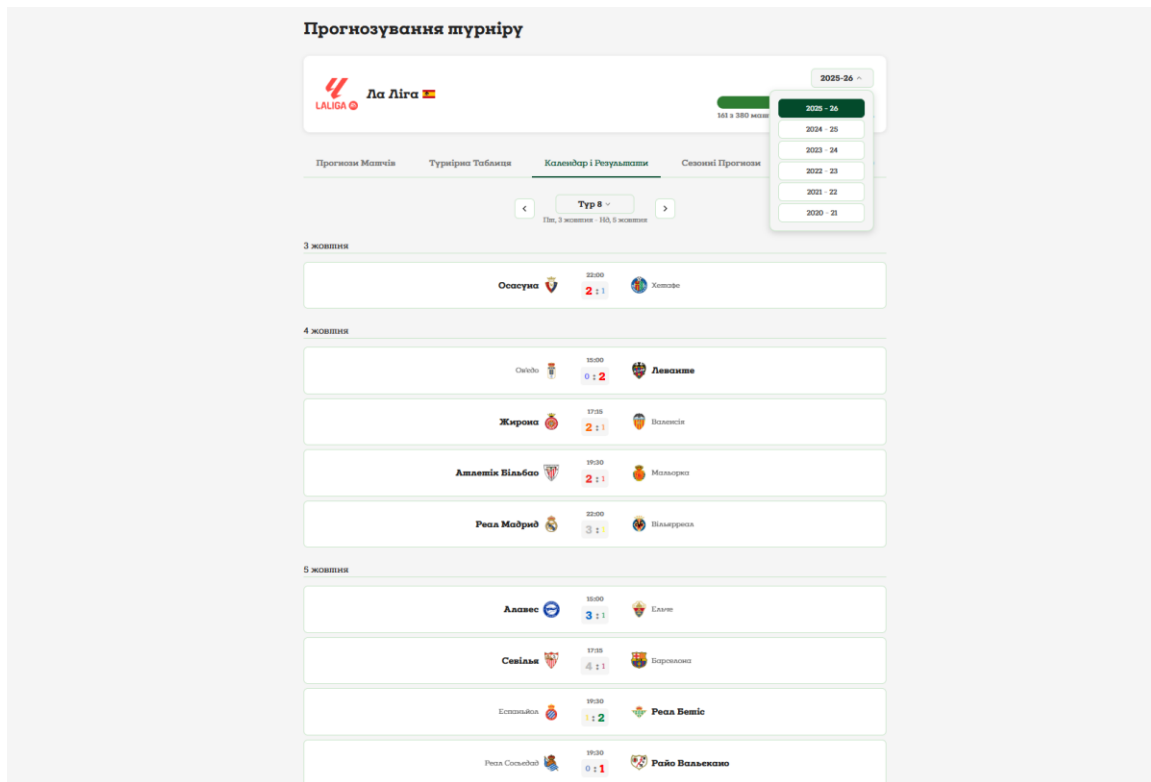


Рис. 3.14 – Сторінка з історичними результатами обраного туру

Висновки до розділу 3

У третьому розділі було розглянуто проектування та реалізацію програмної системи для прогнозування футбольних матчів і підсумкових результатів сезону. Архітектура системи побудована з використанням багатоваріантного підходу, що забезпечує логічне відокремлення рівня взаємодії з користувачем, прикладної логіки обробки даних та обчислювальних модулів прогнозування.

У розділі детально описано структуру серверної частини системи, принципи організації доступу до футбольних даних, механізми кешування та взаємодію компонентів під час формування прогнозів. Окрему увагу приділено послідовності обробки запитів, яка охоплює отримання параметрів через вебінтерфейс, підготовку необхідних характеристик, виконання прогнозування та повернення результатів користувачеві.

Крім того, наведено опис роботи програмного забезпечення з позиції кінцевого користувача, зокрема процес отримання прогнозів для окремих матчів і перегляду сезонних аналітичних оцінок у вебзастосунку. Представлені рішення підтверджують, що розроблена система є практично орієнтованою, масштабованою та придатною для подальшого розвитку.

ВИСНОВКИ

У процесі виконання кваліфікаційної магістерської роботи було розроблено та реалізовано систему аналітики футбольних матчів, призначену для прогнозування результатів окремих поєдинків і формування ймовірнісного прогнозу фінальної турнірної таблиці футбольних ліг. Основна увага в роботі приділялася практичному застосуванню методів машинного навчання для аналізу спортивних даних.

У першому розділі роботи було проаналізовано предметну область футбольної аналітики, визначено основні складності прогнозування результатів матчів і підсумкових позицій команд, а також розглянуто існуючі комерційні сервіси. Аналіз показав, що більшість наявних рішень є непрозорими з точки зору алгоритмів і переважно орієнтовані на ринок ставок. Також було обґрунтовано вибір програмних засобів і технологічного стеку для реалізації системи.

У другому розділі було розроблено комплексні моделі прогнозування результатів футбольних матчів та фінальної турнірної таблиці. На етапі підготовки даних сформовано набір даних на основі відкритих даних платформи Kaggle з детальною статистикою команд та гравців. Розроблено систему інженерії ознак, що включає турнірні показники, метрики поточної форми команд та агреговані характеристики складу. Реалізовано та порівняно три моделі для прогнозування окремих матчів, серед яких модель випадкового лісу продемонструвала найкращу якість ймовірнісного прогнозування за метриками Brier Score та Ranked Probability Score. Для прогнозування фінальної турнірної таблиці розроблено модель на основі рекурентної нейронної мережі LSTM.

У третьому розділі було здійснено проектування та реалізацію програмної системи з багатоваровою архітектурою, що включає рівень представлення у вигляді вебзастосунку, рівень бізнес-логіки для управління даними та обчислювальний рівень для виконання прогнозування. Детально представлено послідовність обробки запитів від отримання параметрів через

вебінтерфейс до формування та відображення результатів прогнозування. Наведено опис роботи програмного забезпечення з позиції кінцевого користувача, включаючи перегляд прогнозів для окремих матчів та аналіз ймовірнісного розподілу позицій у турнірній таблиці.

Отже, розроблена система демонструє ефективність запропонованих рішень та має потенціал для подальшого розвитку, а розроблені моделі та методи прогнозування можуть бути адаптовані для різних футбольних ліг. Система надає користувачам інструмент для отримання ймовірнісних прогнозів результатів матчів та аналізу можливого сценарію закінчення турніру, що дозволяє користувачам приймати обґрунтовані рішення на основі виконаних прогнозів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bubenko O.V., Savitskyi R.S. Using LSTM neural networks to predict the football league standings. Сучасні інформаційні системи та технології: матеріали VII Всеукр. наук.- практ. інтернет-конф. за тематикою «Сучасні комп'ютерні системи та мережі в управлінні» (29 листопада 2024 р., м. Херсон, м. Хмельницький) / за ред. А. А. Григорової. – Херсон: Книжкове видавництво ФОП Вишемирський В. С., 2024. С. 155–156.
2. Bubenko O.V., Savitskyi R.S. Comparison of prognostic rating systems for predicting the final standings of a football league. Комп'ютерні інтелектуальні системи та мережі. Матеріали XVIII Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених (25-27 березня 2025 р.). – Кривий Ріг: Криворізький національний університет, 2025. С. 178–180.
3. Football Predictions | Opta Analyst. URL: <https://theanalyst.com/eu/2024/08/opta-football-predictions>.
4. Kickform – Football Predictions, Stats & Tips | ThePuntersPag. URL: <https://www.thepunterspage.com/kickform/>.
5. Zhong An Chen. A Hybrid Machine Learning Framework for Soccer Match Outcome Prediction: Incorporating Bivariate Poisson Distribution. *ITM Web of Conferences*. 2025. DOI: [10.1051/itmconf/20257003020](https://doi.org/10.1051/itmconf/20257003020).
6. Predicting Football Match Results Using a Poisson Regression Model / K. Loukas et al. *Applied Sciences*. 2024. Vol. 14, no. 16. P. 7230. DOI: [10.3390/app14167230](https://doi.org/10.3390/app14167230).
7. Mansour Y., Schain M. *Machine Learning*. 2021. Vol. 45, no. 2. P. 123–145. DOI: [10.1023/a:1010950718922](https://doi.org/10.1023/a:1010950718922).
8. Banoula M. Introduction to Long Short-Term Memory(LSTM). Simplilearn. URL: <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm>.
9. Rao S. Deep Learning Illustrated, Part 5: Long Short-Term Memory (LSTM). Medium. URL: <https://medium.com/@shreya.rao/deep-learning-illustrated-part-5-long-short-term-memory-lstm-d379fbbc9bc6>.

10. Olah C. Understanding LSTM Networks – colah's blog. Home - colah's blog. URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs>.
11. Rebbouj M., Lotfi S.. Football League Ranking Prediction Using Machine Learning Regression Model. *Annals of Applied Sport Science*. 2023. URL: <https://aassjournal.com/article-1-1226-en.pdf>.
12. Kaggle: Your Machine Learning and Data Science Community. URL: <https://www.kaggle.com/>.
13. Bishrelt Bat-Erdene, Adam Peloguin. Which soccer metrics best predict winning? A data-driven analysis across Europe's top five leagues. *Research Square*. 2025. DOI: [10.21203/rs.3.rs-7682022/v1](https://doi.org/10.21203/rs.3.rs-7682022/v1).
14. Key Performance Indicators Predictive of Success in Soccer: A Comprehensive Analysis of the Greek Soccer League / A. Stafylidis et al. *Journal of Functional Morphology and Kinesiology*. 2024. Vol. 9, no. 2. P. 107. DOI: [10.3390/jfmk9020107](https://doi.org/10.3390/jfmk9020107).
15. Oterhals G., Loras H., Pedersen A. V. Age at Nomination Among Soccer Players Nominated for Major International Individual Awards: A Better Proxy for the Age of Peak Individual Soccer Performance? *Frontiers in Psychology*. 2021. Vol. 12. DOI: [10.3389/fpsyg.2021.661523](https://doi.org/10.3389/fpsyg.2021.661523).
16. Deb S., Dey D. Spatial modeling of shot conversion in soccer to single out goalscoring ability. *Journal of Sports Analytics*. 2019. Vol. 5, no. 4. P. 281–297. DOI: [10.3233/jsa-190281](https://doi.org/10.3233/jsa-190281).
17. Kayanna A. Morgan, Emanuel S. Grant, Eunjin Kim. Survey of Soft Computing Methods to Predict Soccer Win/Loss Probability. *ICISDM*. 2024. DOI: [10.1145/3686397.3686398](https://doi.org/10.1145/3686397.3686398).
18. Elo Ratings: The Ultimate Sports Ranking System - DubStat. URL: <https://dubstat.com/elo-ratings-the-ultimate-sports-ranking-system/>.
19. P. Lucey, A. Bialkowski, M. Monfort, P. Carr, I. Matthews. «Quality vs Quantity»: Improved Shot Prediction in Soccer using Strategic Features from Spatiotemporal Data. *Computer Science*. 2025. URL: <https://www.semanticscholar.org/paper/Quality-vs-Quantity-Improved->

20. Hvattum L. M., Arntzen H. Using ELO ratings for match result prediction in association football. *International Journal of Forecasting*. 2010. Vol. 26, no. 3. P. 460–470. DOI: [10.1016/j.ijforecast.2009.10.002](#).
21. Tianqi Chen, Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. *Association for Computing Machinery*. 2020. DOI: [10.1145/2939672.2939785](#).
22. Berrar D., Lopes P., Dubitzky W. Incorporating domain knowledge in machine learning for soccer outcome prediction. *Machine Learning*. 2018. Vol. 108, no. 1. P. 97–126. DOI: [10.1007/s10994-018-5747-8/](#)
23. Pattern recognition and machine learning. *Choice Reviews Online*. 2023. Vol. 30, no. 06. P. 30–3308. DOI: [10.5860/choice.30-3308](#).
24. Murphy A. H. A New Vector Partition of the Probability Score. *Journal of Applied Meteorology*. 2021. Vol. 12, no. 4. P. 595–600. DOI: [10.1175/1520-0450\(1973\)012%3C0595:anvpot%3E2.0.co;2](#).
25. Gneiting T., Raftery A. E. Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association*. 2020. Vol. 102, no. 477. P. 359–378. DOI: [10.1198/016214506000001437](#).
26. Seemann M., Deursen S. v. Dependency Injection Principles, Practices, and Patterns. Manning Publications, 2019. 552 p.
27. Newman S. Building Microservices. O'Reilly Media, Incorporated, 2021. 2nd ed.. O'Reilly. 612 p.
28. Martin Kleppmann Designing Data Intensive Application. O'Reilly publications, 2020. 613 p.
29. Percival H. J. W. Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices. O'Reilly Media, Incorporated, 2020. 280 p.

30. Albeshar L., Alfayez R. An Observational Study on Flask Web Framework Questions on Stack Overflow (SO). *IET Software*. 2024. Vol. 2024, no. 1. DOI: [10.1049/sfw2/1905538](https://doi.org/10.1049/sfw2/1905538).

ДОДАТКИ

ЛІСТИНГ Класу побудови ознак

```

class FeatureBuilderV2:
    def __init__(self, api_client: IApiClient, feature_order: List[str],
min_matches: int, logger: Optional[ILogger] = None):
        self.api_client = api_client
        self.feature_order = feature_order
        self.min_matches = min_matches
        self.logger = logger

    def build_features(
        self,
        home_team_id: int,
        away_team_id: int,
        league_id: int,
        season: int,
        fixture_date: Optional[str] = None,
    ) -> Dict[str, float]:
        features: Dict[str, float] = {name: self._default_value(name) for
name in self.feature_order}

        fixture_dt = self._parse_date(fixture_date)
        if fixture_dt:
            features["month"] = fixture_dt.month
            features["day_of_week"] = fixture_dt.weekday()
            features["is_midweek"] = 1 if fixture_dt.weekday() in (1, 2,
3) else 0
            features["is_weekend"] = 1 if fixture_dt.weekday() in (5, 6)
else 0
            features["is_winter"] = 1 if fixture_dt.month in (12, 1, 2)
else 0
            features["is_winter_match"] = features["is_winter"]
            features["is_summer_match"] = 1 if fixture_dt.month in (6, 7,
8) else 0
            features["home_year"] = season
            features["away_year"] = season

            standings_data = self.api_client.get_league_standings(league_id,
season, None)
            standings = self._extract_standings_list(standings_data)

            home_stats = self._team_stats(home_team_id, league_id, season,
fixture_dt, standings)
            away_stats = self._team_stats(away_team_id, league_id, season,
fixture_dt, standings)

            if not home_stats or not away_stats:
                return None

            self._fill_team_stats(features, home_stats, prefix="home")
            self._fill_team_stats(features, away_stats, prefix="away")

            features["has_teamstats"] = 1
            self._fill_diffs(features, home_stats, away_stats)

            self._fill_h2h(features, home_team_id, away_team_id)
            self._fill_travel_defaults(features)

```

```

        self._fill_pitch_weather_defaults(features)
        self._fill_transfer_defaults(features)
        self._fill_classification_flags(features)
        self._fill_season_flags(features, home_stats, away_stats)

        ordered = {k: float(features.get(k, self._default_value(k))) for k
in self.feature_order}
        return ordered

    def _team_stats(
        self,
        team_id: int,
        league_id: int,
        season: int,
        fixture_dt: Optional[datetime],
        standings: List[Dict],
    ) -> Optional[Dict[str, float]]:
        try:
            goals_data = self.api_client.get_team_goals(team_id,
league_id, season)
        except Exception:
            goals_data = {}

        played = float(goals_data.get("played", 0) or 0)
        gf = float(goals_data.get("gf", 0) or 0)
        ga = float(goals_data.get("ga", 0) or 0)

        if played <= 0:
            return None

        stand_stats = self._find_standing(standings, team_id)
        rank = float(stand_stats.get("rank", 20)) if stand_stats else 20.0
        points = float(stand_stats.get("points", 0)) if stand_stats else
0.0

        wins = float(stand_stats.get("all", {}).get("win", 0)) if
stand_stats else 0.0
        draws = float(stand_stats.get("all", {}).get("draw", 0)) if
stand_stats else 0.0
        losses = float(stand_stats.get("all", {}).get("lose", 0)) if
stand_stats else 0.0

        form_stats = self._form_stats(team_id, league_id, season)

        games_for_rates = max(played, 1.0)
        gf_pg = gf / games_for_rates
        ga_pg = ga / games_for_rates
        goal_diff = gf - ga
        goal_diff_pg = goal_diff / games_for_rates
        points_per_game = points / games_for_rates
        win_rate = wins / games_for_rates
        draw_rate = draws / games_for_rates
        loss_rate = losses / games_for_rates

        return {
            "played": played,
            "gf": gf,
            "ga": ga,
            "rank": rank,
            "points": points,

```

```

        "wins": wins,
        "draws": draws,
        "losses": losses,
        "gf_pg": gf_pg,
        "ga_pg": ga_pg,
        "goal_diff": goal_diff,
        "goal_diff_pg": goal_diff_pg,
        "points_per_game": points_per_game,
        "win_rate": win_rate,
        "draw_rate": draw_rate,
        "loss_rate": loss_rate,
        **form_stats,
        "days_since_last": self._days_since_last_match(team_id,
league_id, season, fixture_dt),
    }

    def _fill_team_stats(self, features: Dict[str, float], stats:
Dict[str, float], prefix: str):
        features[f"{prefix}_gamesPlayed"] = stats["played"]
        features[f"{prefix}_wins"] = stats["wins"]
        features[f"{prefix}_ties"] = stats["draws"]
        features[f"{prefix}_losses"] = stats["losses"]
        features[f"{prefix}_points"] = stats["points"]
        features[f"{prefix}_gf"] = stats["gf"]
        features[f"{prefix}_ga"] = stats["ga"]
        features[f"{prefix}_teamRank"] = stats["rank"]
        features[f"{prefix}_gf_per_game_calc"] = stats["gf_pg"]
        features[f"{prefix}_ga_per_game_calc"] = stats["ga_pg"]
        features[f"{prefix}_goal_diff_calc"] = stats["goal_diff"]
        features[f"{prefix}_goal_diff_per_game_calc"] =
stats["goal_diff_pg"]
        features[f"{prefix}_points_per_game_calc"] =
stats["points_per_game"]
        features[f"{prefix}_win_rate_calc"] = stats["win_rate"]
        features[f"{prefix}_draw_rate_calc"] = stats["draw_rate"]
        features[f"{prefix}_loss_rate_calc"] = stats["loss_rate"]
        features[f"{prefix}_form_points_calc"] = stats["form_points"]
        features[f"{prefix}_form_wins_calc"] = stats["form_wins"]
        features[f"{prefix}_form_draws_calc"] = stats["form_draws"]
        features[f"{prefix}_form_losses_calc"] = stats["form_losses"]
        features[f"{prefix}_clean_sheet_rate_calc"] = 0.0
        features[f"{prefix}_clean_sheet"] = 0.0
        features[f"{prefix}_form_last_match"] = stats["form_last"]
        features[f"{prefix}_deductions"] = 0.0
        features[f"{prefix}_xg_proxy"] = stats["gf_pg"]
        features[f"{prefix}_days_since_last_match"] =
stats["days_since_last"]
        features[f"{prefix}_lineup_changes_calc"] = 0.0

    def _fill_diffs(self, features: Dict[str, float], home: Dict[str,
float], away: Dict[str, float]):
        features["rank_diff_calc"] = home["rank"] - away["rank"]
        features["points_diff_calc"] = home["points"] - away["points"]
        features["points_per_game_diff_calc"] = home["points_per_game"] -
away["points_per_game"]
        features["win_rate_diff_calc"] = home["win_rate"] -
away["win_rate"]
        features["gf_per_game_diff_calc"] = home["gf_pg"] - away["gf_pg"]
        features["ga_per_game_diff_calc"] = home["ga_pg"] - away["ga_pg"]

```

```

        features["form_diff_calc"] = home["form_points"] -
away["form_points"]
        features["attack_vs_defense_calc"] = home["gf_pg"] - away["ga_pg"]
        features["defense_vs_attack_calc"] = home["ga_pg"] - away["gf_pg"]
        features["xg_diff_calc"] = features.get("home_xg_proxy", 0.0) -
features.get("away_xg_proxy", 0.0)
        features["height_diff_calc"] = 0.0
        features["elo_diff_calc"] = features.get("home_elo_before", 1500)
- features.get("away_elo_before", 1500)
        features["elo_home_expected_calc"] = self._elo_expected(
            features.get("home_elo_before", 1500),
features.get("away_elo_before", 1500)
        )

    def _elo_expected(self, elo_home: float, elo_away: float) -> float:
        return 1 / (1 + 10 ** ((elo_away - elo_home) / 400))

    def _fill_h2h(self, features: Dict[str, float], home_team_id: int,
away_team_id: int):
        try:
            h2h_fixtures = self.api_client.get_head_to_head(home_team_id,
away_team_id)
        except Exception:
            h2h_fixtures = []

        if not h2h_fixtures:
            return

        total = min(len(h2h_fixtures), 50)
        wins_home = draws = wins_away = goals_total = btts_count = 0

        for fixture in h2h_fixtures[:total]:
            score = fixture.get("score", {})
            fulltime = score.get("fulltime", {})
            h = fulltime.get("home")
            a = fulltime.get("away")
            if h is None or a is None:
                continue
            goals_total += h + a
            if h > a:
                wins_home += 1
            elif h == a:
                draws += 1
            else:
                wins_away += 1
            if h > 0 and a > 0:
                btts_count += 1

        if total > 0:
            features["h2h_matches_count"] = total
            features["h2h_home_win_pct"] = wins_home / total
            features["h2h_draw_pct"] = draws / total
            features["h2h_away_win_pct"] = wins_away / total
            features["h2h_avg_total_goals"] = goals_total / total
            features["h2h_btts_rate"] = btts_count / total

```

Лістинг отримання даних із зовнішнього API

```

def get_league_fixtures(self, league_id: int, selected_date: str = None) -
> Dict[str, List[Dict]]:
    if selected_date:
        date_range = {"from": selected_date, "to": selected_date}
    else:
        date_range = self._get_date_range()

    season = SeasonCalculator.get_season_for_fixtures()
    league_name = self._get_league_name_by_id(league_id)

    fixtures = self._fetch_league_fixtures(league_id, season, date_range)

    return {league_name: fixtures}

def fetch_fixtures_for(
    self, team_id: int, league_id: int, season: int, from_date: str,
    to_date: str
) -> List[int]:
    cached = self.cache_service.get_cached_data("team_fixtures_filtered",
team_id=team_id, league_id=league_id, season=season, from_date=from_date,
to_date=to_date)
    if cached:
        if self.logger:
            self.logger.debug(f"Cache hit for filtered fixtures:
team={team_id}, league={league_id}, season={season}")
        return cached

    params = {
        "team": team_id,
        "league": league_id,
        "season": season,
        "from": from_date,
        "to": to_date,
        "status": ApiConstants.FIXTURE_STATUS_FULL_TIME,
    }
    response = self._make_request(f"{self.base_url}/fixtures", params)

    result = FixtureMapper.extract_fixture_ids(response)
    if result:
        self.cache_service.cache_data("team_fixtures_filtered", result,
team_id=team_id, league_id=league_id, season=season, from_date=from_date,
to_date=to_date)

    return result

def get_fixture_statistics(self, fixture_id: int, team_id: int) ->
List[Dict]:
    cached = self.cache_service.get_cached_data("fixture_stats",
fixture_id=fixture_id, team_id=team_id)
    if cached:
        if self.logger:
            self.logger.debug(f"Cache hit for fixture statistics:
fixture={fixture_id}, team={team_id}")
        return cached

```

```

        params = {"fixture": fixture_id, "team": team_id}
        response = self._make_request(f"{self.base_url}/fixtures/statistics",
params)

        result = FixtureMapper.extract_statistics(response)
        if result:
            self.cache_service.cache_data("fixture_stats", result,
fixture_id=fixture_id, team_id=team_id)

        return result

    def get_team_goals(self, team_id: int, league_id: int, season: int) ->
Dict:
        cached = self.cache_service.get_cached_data("team_goals",
team_id=team_id, league_id=league_id, season=season)
        if cached:
            if self.logger:
                self.logger.debug(f"Cache hit for team_goals: team={team_id},
league={league_id}, season={season}")
            return cached

        params = {"team": team_id, "league": league_id, "season": season}
        response = self._make_request(f"{self.base_url}/teams/statistics",
params)

        result = FixtureMapper.extract_goals_data(response)
        if result:
            self.cache_service.cache_data("team_goals", result,
team_id=team_id, league_id=league_id, season=season)

        return result

```

Лістинг класу, що інкапсулює роботу з моделями прогнозування
футбольних матчів.

```

from pathlib import Path
from typing import Any, Dict, Optional, Tuple

import joblib
import numpy as np
from scipy.stats import poisson

class ModelBundleRfPoisson:
    def __init__(self, base_dir: str):
        self.base_dir = Path(base_dir)

        self.feature_cols = self._load_json("feature_cols_v2.json")
        self.scaler = self._load_joblib("scaler_v2.pkl")

        old_model_dir = self.base_dir.parent
        old_features_info = self._load_json_from_path(old_model_dir /
"features_info.json")
        self.old_outcome_features = old_features_info["feature_names"]
        self.rf_outcome_old = self._load_joblib_from_path(old_model_dir /
"rf_outcome.joblib")

        self.rf_btts = self._load_joblib("rf_btts_v2.pkl")

        self.poisson_xgb = self._load_joblib("poisson_xgb_v2.pkl")

    def predict(self, features: Dict[str, float]) -> Dict[str, Any]:
        X = np.array([[features.get(col, 0.0) for col in
self.feature_cols]], dtype=float)
        X_scaled = self._safe_scale(X)

        X_old = np.array([[features.get(col, 0.0) for col in
self.old_outcome_features]], dtype=float)
        outcome_proba =
self._normalize_or_fallback(self.rf_outcome_old.predict_proba(X_old)[0])

        btts_yes = float(self.rf_btts.predict_proba(X_scaled)[0, 1])

        lambda_home, lambda_away = self.poisson_xgb.predict(X_scaled)[0]
        lambda_home = float(max(lambda_home, 0.0))
        lambda_away = float(max(lambda_away, 0.0))

        matrix, total = self._poisson_matrix(lambda_home, lambda_away,
max_goals=7)
        matrix = matrix / total if total > 0 else matrix

        over15 = self._poisson_over_prob(matrix, 1.5)
        over25 = self._poisson_over_prob(matrix, 2.5)
        over35 = self._poisson_over_prob(matrix, 3.5)

        labels = ["H", "D", "A"]
        predicted_outcome = labels[int(np.argmax(outcome_proba))]

```

```

        return {
            "outcome": {
                "homeWin": round(float(outcome_proba[0]), 3),
                "draw": round(float(outcome_proba[1]), 3),
                "awayWin": round(float(outcome_proba[2]),
3),
            },
            "btts": {"probability": round(btts_yes, 3), "predicted":
btts_yes > 0.5},
            "overUnder": {
                "over15": round(over15, 3),
                "over25": round(over25, 3),
                "over35": round(over35, 3),
            },
            "meta": {"modelVersion": "rf+poisson"},
        }

    def _poisson_matrix(self, lambda_home: float, lambda_away: float,
max_goals: int) -> Tuple[np.ndarray, float]:
        matrix = np.zeros((max_goals + 1, max_goals + 1))
        total = 0.0
        for h in range(max_goals + 1):
            for a in range(max_goals + 1):
                p = float(poisson.pmf(h, lambda_home) * poisson.pmf(a,
lambda_away))
                matrix[h, a] = p
                total += p
        return matrix, total

    float:
    def _poisson_over_prob(self, matrix: np.ndarray, threshold: float) ->
float:
        prob = 0.0
        for h in range(matrix.shape[0]):
            for a in range(matrix.shape[1]):
                if h + a > threshold:
                    prob += float(matrix[h, a])
        return prob

    def _safe_scale(self, X: np.ndarray) -> np.ndarray:
        try:
            return self.scaler.transform(X)
        except Exception:
            return X

    def _normalize_or_fallback(self, probs: np.ndarray) -> np.ndarray:
        s = float(np.sum(probs))
        if s <= 0 or np.isnan(s):
            return np.array([0.33, 0.34, 0.33], dtype=float)
        return probs / s

    def _load_json(self, filename: str) -> Any:
        path = self.base_dir / filename
        with open(path, "r", encoding="utf-8") as f:
            return __import__("json").load(f)

    def _load_json_from_path(self, path: Path) -> Any:
        with open(path, "r", encoding="utf-8") as f:
            return __import__("json").load(f)

```

```
def _load_joblib(self, filename: str):  
    return joblib.load(self.base_dir / filename)  
  
def _load_joblib_from_path(self, path: Path):  
    return joblib.load(path)
```