МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПЗВО «МІЖНАРОДНИЙ КЛАСИЧНИЙ УНІВЕРСИТЕТ
імені ПИЛИПА ОРЛИКА»
Економіко-технологічний факультет
Кафедра інженерних технологій

Кваліфікаційна робота
на здобуття освітнього ступеня магістра
за освітньою програмою «Комп'ютерна інженерія»
зі спеціальності 123 «Комп'ютерна інженерія»
на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПРОГНОЗУВАННЯ СПОРТИВНИХ ПОДІЙ»**

Виконала:
здобувач ІІ курсу, групи КІ -20-24
Скрипкару Анна Олегівна

Керівник:
к.т.н., доцент кафедри інженерних технологій
Гайша Олександр Олександрович

Миколаїв – 2024

РЕФЕРАТ

Кваліфікаційна робота присвячена вирішенню проблеми розробки удосконаленої технології прогнозування результатів спортивних подій.

Актуальність теми. Прогнозування є важливою задачею сучасної галузі ІТ, що має досить серйозне математичне підґрунтя. Якісне вирішення цієї задачі дозволяє підвищити якість процесу тренування та підготовки до змагань, а у випадку виконання ставок на тоталізаторі, дозволяє підвищити ефективність всієї гри.

Мета і задачі дослідження.

Метою роботи є підвищення якості прогнозування результатів спортивних подій, що може бути виконано шляхом удосконалення однієї із існуючих і гарно себе зарекомендувавших методик.

Для досягнення мети потрібно вирішити наступні задачі дослідження:

- проаналізувати сучасну науково-технічну літературу та електронні джерела на предмет пошуку сучасних ефективних способів прогнозування;

- обґрунтовано обрати один із ефективних способів, що допускає можливість покращення в рамках виконання магістерської роботи;

- внести нові характеристики у процес прогнозування, удосконаливши обраний раніше алгоритм;

- вибрати технології та засоби розробки;

- здійснити програмну реалізацію удосконаленого алгоритму;

- виконати тестування та оцінити ефективність всієї роботи.

Об’єкт дослідження – прогнозування результатів спортивних подій.

Предмет дослідження – методика прогнозування результатів спортивних подій на основі кваліметричного підходу.

Методи дослідження: методи математичного моделювання, математичної статистики, структурний підхід до програмування.

Наукова новизна одержаних результатів полягає у наступному:

- здійснено аналітичний огляд науково-технічних джерел, присвячених прогнозуванню результатів спортивних подій та встановлено, що найкраще

для удосконалення в умовах обмеженого часу та ресурсів підходить метод, заснований на кваліметричних підходах;

- вперше запропоновано окрім параметрів спортивної події, що прогнозується, брати також до уваги параметри і результати попередніх спортивних подій, пов’язаних із цим учасником, що дозволяє залучити нову інформацію і підвищити якість прогнозування.

Практичне значення одержаних результатів полягає у тому, що стає можливим безпосередній розрахунок рейтингу учасників спортивних змагань підвищеної точності, в результаті чого вищими є шанси отримання грошової винагороди при виконанні ставок, а також стає можливим більш якісна спортивна підготовка самих учасників (за умови наявності достовірного прогнозу події).

Структура та обсяг роботи. Дипломна робота складається із вступу, 4 розділів, висновків, переліку використаних джерел на 32 позиції та 3 додатків з висхідними текстами розробленого програмного забезпечення. Загальний обсяг роботи складає 80 сторінок, з них 73 сторінок основного тексту та 7 сторінок додатків. У роботі 12 рисунків та 1 таблиця.

Ключові слова: прогнозування, матч, кваліметрія, нейронні мережі, нечіткі множини.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ASP	Active Server Pages
CSS	Cascading Style Sheet
CSV	Comma Separated Values
GUI	Graphics User Interface
HTML	HyperText Markup Language
JSP	Java Server Pages
PC	Personal Computer
PHP	Personal Home Page, **P**HP **H**ypertext **P**reprocessor
SQL	Structured Queries Language
WWW	World Wide Web
XML	eXtensible Markup Language
БД	База даних
ВНЗ	Вищий навчальний заклад
ЗМІ	Засоби масової інформації
ІТ	Інформаційні технології
ОО	Об’єктно-орієнтований (-на, -не)
ООП	Об’єктно-орієнтоване програмування
ПЗ	Програмне забезпечення
ПК	Персональний комп’ютер
СУБД	Система управління базами даних
ТНМ	Теорія нечітких множин
ШНМ	Штучна нейронна мережа

ВСТУП

З моменту свого виникнення сучасні інформаційні технології значно розширили спектр галузей свого практичного застосування. Одним із магістральних напрямків їх розвитку полягає у розробленні методів та засобів прогнозування в самому широкому сенсі цього слова.

Дійсно, у будь-якому процесі (що розглядається не у свій в початковий момент) є певна передісторія, хід його виконання. Ця інформація завжди може бути представлена у цифровій формі (навіть при словесному описі у якісних категоріях дослідник може застосувати механізм теорії нечітких множин для формалізації проблеми «до числа»), і за допомогою спеціальних методів, які в цілому можуть бути віднесені до галузі ІТ (або надзвичайно тісно пов’язані із нею, зважаючи на широке використання комп’ютерної техніки у будь-якому із них), у цифровій же формі можна отримати і прогнозні значення для даного процесу, тобто передбачити його хід у найближчому (зазвичай, із більшим ступенем достовірності) або і більш віддаленому майбутньому.

Прогнозування є чи не найпершою задачею, для якої взагалі створювалася комп’ютерна техніка (початково, перші електронно-обчислювальні машини – ЕОМ – розроблялися переважно для розрахунку процесу польоту бойових ракет, який, при розгляді цього процесу як стохастичного – з урахуванням різноманітних випадкових збурень, також може бути віднесений до прогнозів). На сьогоднішній день методи прогнозування активно використовуються в економіці, соціології, політиці, біології, метеорології, медицині, природничих науках і т.п. Однією із галузей застосування також є і спорт.

У спорті існує ціла низка причин, через які прогнозування результатів спортивних подій є актуальною і вкрай необхідною задачею, що потребує розроблення удосконалених (або нових) ефективних методів вирішення. В першу чергу, реальна оцінка можливого результату події може підвищити ефективність планування всього процесу підготовки, визначатиме тактику та

стратегію під час змагання, і т.п. Наприклад, футбольні команди під час матчів дуже високого рівня, але коли результат змагання добре прогнозований (наприклад, коли ослаблена травмами команда виступає проти суперника того ж рівня, але такого, що знаходиться на піку), може дозволити собі ризиковані рішення: спробувати нову схему гри, дати спробувати нові амплуа для досвідчених гравців, випустити на поле новобранців, аби дати їм можливість «відчути серйозну гру», і т.п. Так само і в боксі: якщо прогноз поєдинку явно схиляється в одну із сторін, інша може запросити переніс на наступний період для створення можливості кращої підготовки, і т.д. В усіх видах спорту наявність гарного прогнозу є важливим фактором, що може бути використаний для підвищення шансів учасників змагань на перемогу.

Однак, існує і інша не менш значима група осіб (або навіть і більш чисельна), для яких важливим є якісне прогнозування результату – це вболівальники, що роблять ставки в букмекерських конторах, на тоталізаторах, беруть участь у спортивних лотереях. Тут якість прогнозу прямим чином впливає на грошову сторону питання, тому використання ефективних методів також є надзвичайно важливим і значущим.

Зважаючи на наведені факти, розробка (удосконалення) ефективних методів прогнозування результатів спортивних подій є актуальною науково-технічною задачею, яку доцільно вирішувати, зокрема, в рамках написання магістерської роботи.

РОЗДІЛ 1. Аналіз сучасного стану використання інформаційних технологій у сфері прогнозування результатів спортивних подій

1.1. Суть та особливості задачі прогнозування в цілому, та зокрема у сфері спорту

Прогнозування в самому загальному сенсі полягає у виробленні майбутнього стану системи чи процесу $\vec{x}_{n+1}$ на основі наявних відомостей про попередні стани $\vec{x}_n$, $\vec{x}_{n-1}$, $\vec{x}_{n-2}$, …, $\vec{x}_k$ (причому сюди включаються як вхідні, так і вихідні змінні для кожного стану) та вхідну інформацію про наступний стан, який прогнозується (наприклад, вектор його вхідних параметрів $\vec{u}_{n+1}$):

$$\vec{x}_{n+1} = f\left(\vec{x}_n, \vec{x}_{n-1}, \vec{x}_{n-2}, ..., \vec{x}_k, \vec{u}_{n+1}\right) \quad (1.1)$$

де $\vec{x}_i$ - набір (вектор) параметрів, якими описується *i*-тий стан системи (процесу);

$\vec{x}_{n+1}$ – стан, що прогнозується, або, коротко, прогноз. В широкому сенсі він включає набір вхідних величин $\vec{u}_{n+1}$, що описують даний стан, та вихідних $\vec{v}_{n+1}$, які власне і є прогнозом, якщо говорити більш уточнено. Можна сказати, що:

$$\vec{x}_{n+1} = \vec{u}_{n+1} \cup \vec{v}_{n+1}; \quad (1.2)$$

$\vec{x}_n$, $\vec{x}_{n-1}$, $\vec{x}_{n-2}$, …, $\vec{x}_k$ – попередні стани, повна інформація про які є відомою (а саме, для кожного відомий набір вхідних змінних та вихідних змінних окремо). Кількість таких станів позначимо $p = n - k + 1$;

$\vec{x}_k$ – найбільш віддалений у минулому часі стан, що ще враховується для побудови прогнозу;

$\vec{u}_{n+1}$ - набір значень для вхідних змінних, на основі яких обчислюватиметься набір вихідних даних $\vec{v}_{n+1}$, тобто формуватиметься набір $\vec{x}_{n+1}$ в цілому.

Характер інформації, яка являє собою прогноз $\vec{x}_{n+1}$, відповідає суті інформації $\vec{x}_n$, $\vec{x}_{n-1}$, $\vec{x}_{n-2}$, …, що є відомою про попередні стани.

Наприклад, якщо описується процес виготовлення деталі складної геометрії із певною кількістю отворів x_1 та паяних з'єднань x_2 (причому, зазвичай, x_1 і x_2 є досить великими числами), то час виготовлення деталі для даного конкретного працівника складає x_3, а точність виготовлення (наприклад, по показнику маси) оцінюється відносним відхиленням x_4 (наприклад, у відсотках). В цілому якийсь *i*-тий стан системи можна описати четвіркою чисел:

$$\vec{x}_i = \{x_{i1}, x_{i2}, x_{i3}, x_{i4}\},$$

де весь вектор можна розбити на дві складові: $\vec{u}_i = \{x_{i1}, x_{i2}\}$ та $\vec{v}_i = \{x_{i3}, x_{i4}\}$, з яких перша уособлює вхідні параметри, а друга – вихідні. Власне прогнозом, є набір $\vec{v}_{n+1} = \{x_{(n+1)3}, x_{(n+1)4}\}$, але так само (у більш широкому сенсі) можна назвати і весь набір параметрів $\vec{x}_{n+1} = \{x_{(n+1)1}, x_{(n+1)2}, x_{(n+1)3}, x_{(n+1)4}\}$, тобто в т.ч. і вхідних, якими описується прогнозований стан системи або процесу.

В загальному випадку кількість координат вектора, що описує *i*-тий стан може бути довільним, тому введемо для нього власне позначення *m*:

$$\vec{x}_i = \{x_{i1}, x_{i2}, ..., x_{im}\} = \{x_{ij} \mid j = 1..m\}. \qquad (1.3)$$

Розвиваючи (1.2) на усі стани $\vec{x}_i$, що беруть участь у розрахунках, формулу (1.3) можна подати ще і наступним чином:

$$\vec{x}_i = \vec{u}_i \cup \vec{v}_i = \{u_{ij} \mid j = 1..m_u\} \cup \{v_{ij} \mid j = 1..m_v\}, \qquad (1.4)$$

де m_u – кількість вхідних змінних, що описують стан системи;

m_v – кількість вихідних змінних, що описують стан системи (власне, це кількість величин, що мають бути передбачені, якщо говорити про скалярні величини, а не узагальнено про векторну величину $\vec{v}_i$).

Відмітимо, що

$$m = m_u + m_v. \qquad (1.5)$$

Елементи множин (векторів) $\vec{u}_i$ та $\vec{v}_i$ співпадають з деякими складовими вектора $\vec{x}_i$ (оскільки, власне, він утворюється об'єднанням цих двох векторів), що можна відобразити у наступних співвідношеннях:

$$u_{ij} \equiv x_{ij}\,|_{j=1..m_u}\ ;\ v_{ij} \equiv x_{i(m_u+j)}\,|_{j=1..m_v}. \qquad (1.6)$$

Якщо стан системи описується у лінгвістичних термінах, наприклад, «імовірність перемоги боксера *A* висока, якщо його опонент середнього зросту», то і прогноз буде виконуватися у таких термінах.

Таким чином, структура та суть прогнозної інформації повністю відповідає параметрам інформації про попередні (відомі) стани системи.

Також обов'язково слід пам'ятати, що для успішного вирішення задачі прогнозування потрібним є набір даних не лише про попередні стани системи чи процесу, а й набір вхідних параметрів, що супроводжуватимуть той стан, що прогнозується. Аби підкреслити цю тезу, можна переписати (1.1) в дещо іншій формі з більш докладним урахуванням (1.2):

$$\vec{x}_{n+1} = f\left(\mathbf{X}, \vec{u}_{n+1}\right), \qquad (1.7)$$

де **X** – матриця, що містить інформацію про усі попередні стани системи, що ураховуються при прогнозуванні. Її можна записати наступним чином:

$$\begin{gathered}\mathbf{X} = \left\{\vec{x}_n, \vec{x}_{n-1}, ..., \vec{x}_k\right\} = \left\{\vec{x}_i \mid i = n..k\right\} = \left\{\left\{x_{ij} \mid j = 1..m\right\} \mid i = n..k\right\} = \\ = \left\{\left\{u_{ij} \mid j = 1..m_u\right\} \cup \left\{v_{ij} \mid j = 1..m_v\right\} \mid i = n..k\right\}\end{gathered} \qquad (1.8)$$

Матрицю **X** можна також описати у текстовій формі наступним чином: вона складається із рядків, кожен з яких є набором усіх координат вектора $\vec{x}_i$, тобто описує *i*-тий стан системи, який, відповідно, ураховується в прогнозній моделі. Перші m_u елементів кожного рядка являють собою значення вхідних параметрів для стану, який описується даним рядком. Наступні m_v елементів (тобто до самого кінця рядка, враховуючи (1.5)) є значеннями вихідних параметрів, що описували стан, який відповідає даному рядку.

Аналіз (1.7) дозволяє ввести просту, але ефективну класифікацію методів прогнозування за наявністю того, чи іншого аргументу в рівнянні (1.7). Всього можна виділити три випадки схем прогнозування:

1) коли враховується тільки інформація **X** про попередні стани, але не враховується наявна інформація $\vec{u}_{n+1}$ про вхідні параметри стану, який буде прогнозуватися:

$$\vec{x}_{n+1} = f(\mathbf{X}). \qquad (1.9)$$

Такий підхід означає вибудовування певного тренду, закону, по якому міняються у часі (мається на увазі дискретний час, коли кожен новий момент просто відповідає утворенню нового стану системи) параметри стану. Підхід добре працює за відсутності суттєвих несподіваних збурюючих факторів.

Наприклад, у країні з економікою, що активно розвивається, курс національної валюти зростає на кілька десятих долей відсотка щомісяця у порівнянні з курсом долара США. На основі такої інформації вже можна говорити про прогнозне значення курсу валюти через місяць, два, три, але все це дасть більш-менш точний прогноз лише за умови відсутності несподіваних глобальних потрясінь, таких як сильний землетрус, збройна агресія сусідньої держави, банкротство національних компаній чи найбільших корпорацій, і т.п.

Більш витончений приклад можна навести якраз з галузі прогнозування результатів спортивних подій. Шляхом ретельних спостережень можна встановити, що команда «А» у своїй Національній лізі (де більшість команд мають приблизно однаковий рівень) виграє 2 матчі поспіль, а потім майже завжди програє один, після чого все повторюється. Така ситуація цілком природно може бути обумовлена тим, що після програшу більша частина команди іде в бар і потім не може стати до тренувань кілька днів, накопичуючи сили. Під час же звичайних тренувань, навантаження, що дає тренер, є занадто важким для даної команди і гравці поступово перегоряють фізично, не маючи достатньо часу на відновлення. Запасу сил більшості гравців вистачає на термін між двома іграми, а на третю сил вже не

залишається. Знаючи таку інформацію, можна досить точно зробити прогноз, за умови, що сила суперників у лізі є приблизно однаковою. При цьому результат буде отримано лише на основі **X**, без урахування інших, очевидно відомих, обставин $\vec{u}_{n+1}$ прогнозного поєдинку.

Для виявлення трендів використовують методи математичної статистики типу регресійного аналізу, сплайн-апроксимацію, перетворення Фур'є із подальшою фільтрацією, і т.п.

2) у наступній схемі організації прогнозної моделі враховуються тільки відомі обставини $\vec{u}_{n+1}$ майбутнього стану, але без залучення даних про історію попередніх станів **X**:

$$\vec{x}_{n+1} = f(\vec{u}_{n+1}). \qquad (1.10)$$

У галузі прогнозування результатів спортивних подій така ситуація є досить поширеною, а такі моделі називають кваліметричними. При цьому враховуються поточні значення різноманітних суттєвих показників, важливих для предметної галузі, що розглядається. Ці значення осереднюються, зважуються, згортаються і в результаті, звичайно, отримують якийсь один показник, за величиною якого можна зробити висновок про результат прогнозування.

З математичної точки зору для цього випадку застосовуються різноманітні методи згортки, нормалізації, методи експертного аналізу, і т.д.

3) нарешті, об'єднати переваги обох підходів та позбутися їх недоліків можна шляхом реалізації (1.7), де враховуються обидва типи інформації, наявної в системі.

1.2. Аналіз науково-технічних джерел, присвячених проблемі прогнозування спортивних подій та суміжних галузей

Проблемі прогнозування результатів спортивних подій присвячено чимало наукових джерел, причому (як це не дивно) переважна частка із них написана не для допомоги спортсменам, а з точки зору особливостей виконання ставок та гри на тоталізаторі (що показує серйозне ставлення

науковців до проблем азартних ігор та близьких до них процесів). Цікаво, що для вирішення цієї проблеми науковці залучають досить різні математичні підходи (іноді – діаметрально протилежні), причому усі вони декларують гарні результати прогнозування. Розглянемо деякі з них.

Укрупнено усі методи прогнозування, зокрема і результатів спортивних подій, можна поділити на:

- статистичні методи;

- методи з використанням нейронних мереж;

- застосування нечіткої логіки;

- інші методи інтелектуального аналізу даних, що використовують більш екзотичні алгоритми, ніж вищенаведені.

Значна частка наукових робіт у цій галузі присвячена традиційним статистичним методам з урахуванням кваліметричних показників спортивних команд (учасників змагань). Так, наприклад, в [1-5] різними авторами розглядається практично один і той самий підхід, при якому послідовність дій для виконання прогнозу є наступною:

- обирається група показників, які, на думку авторів, є важливими для суті поставленої задачі;

- величина кожного показника нормалізується, зазвичай, просто на 1 (наприклад, шляхом простого ділення на якесь реперне максимальне значення цього показника);

- для кожного показника обирається його «вага» для даної предметної галузі (очевидно, що для прогнозування різних видів спорту, ваги будуть різнитися, адже, наприклад, травмування одного члена команди може бути мало суттєвим у футболі, але критичним – у бобслеї);

- обчислюється підсумкове, результуюче значення якогось інтегрального показника, на основі якого за певними правилами (зазвичай, це прості інтервальні нерівності, одній з яких і відповідає отриманий підсумковий результат розрахунків) робиться висновок про кінцевий результат спортивного змагання.

При такому підході не враховується інформація про параметри попередніх станів процесу, тобто даний підхід в чистому вигляді можемо віднести до типу (1.10). Моделі, що відповідають такому ходу дій можна назвати кваліметричними, скоринговими (від англ. score – очки, бали), зваженим сумуванням, адитивними, зважено-адитивними, і т.д.

Окремим класом таких систем, які, в принципі, можуть застосовуватися і для статистичних підходів, є рішення на базі нейронних мереж, яким присвячена чи не найбільша відносна частка усіх наукових джерел останніх років по спортивному прогнозуванню, наприклад [6-10]. Така ситуація у великій мірі обумовлена певним «бумом», що спостерігається в галузі штучного інтелекту та суміжних напрямках, навколо поняття нейронної мережі. Цей об’єкт широко застосовується для задач розпізнавання, класифікації, апроксимації, а, зважаючи на це – і для прогнозування.

Штучною нейронною мережею (далі – ШНМ), як відомо, називають систему, яка, приймаючи на вхід вектор вхідних величин, за складними (нелінійними) внутрішніми законами виробляє вектор вихідних величин, що є більш-менш задовільним розв’язком поставленої задачі.

Елементарною складовою ШНМ є один нейрон – рис. 1.1. Це об’єкт, що має певну кількість входів (на рисунку – n входів), які називають дендритами та один вихід, що називається аксоном.

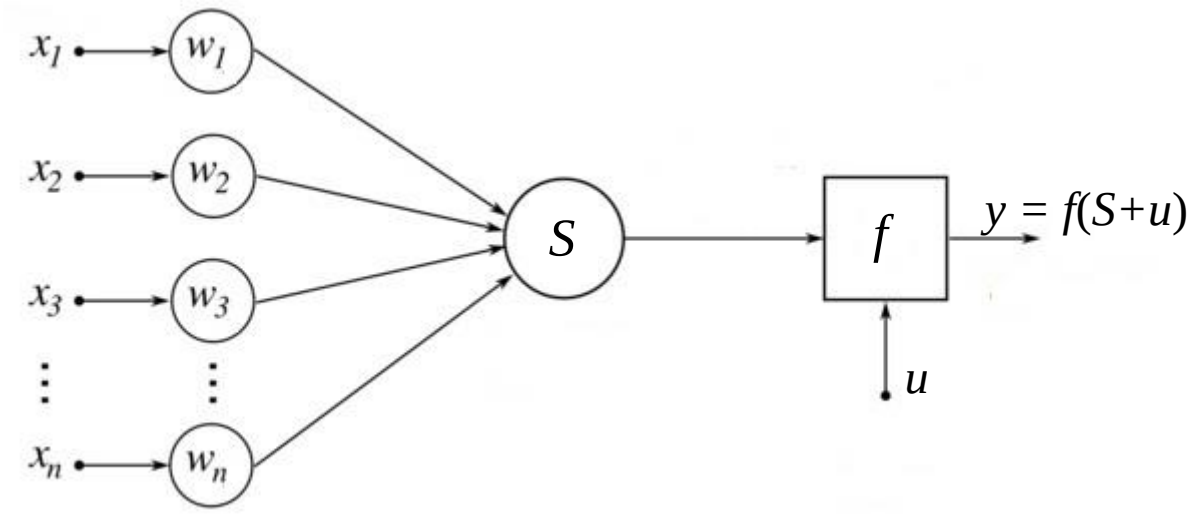

Рис. 1.1 Структура штучного нейрону.

На входи нейрону подаються вхідні сигнали x_1, x_2, …, x_n, що є виходами попередніх нейронів, розташованих раніше по ходу проходження сигналу. Кожен вхідний сигнал x_i множиться на певне число w_i, що називають вагою (або синаптичною вагою) відповідного входу. Усі таким чином зважені вхідні сигнали подаються на суматор, який виробляє зважену суму S виду:

$$S = \sum_{i=1}^{n} w_i x_i \,. \tag{1.11}$$

До цієї зваженої суми вхідних сигналів може додаватися зсув або зміщення u (у поширеному частинному випадку зміщення відсутнє, тобто $u = 0$) і від цієї суми виробляється певна функція f, що називається активаційною; так і формується вихідний сигнал нейрону y, що передається далі по нейронній мережі:

$$y = f(S + u). \tag{1.12}$$

Відповідно комбінуючи (1.11) та (1.12) маємо функцію, що здійснює штучний нейрон:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + u \right). \tag{1.13}$$

При аналізі такого способу завдання функції, що здійснює нейрон, бачимо, що важливу роль для кожного нейрона відіграють три речі (особливо перші дві):

- набір конкретних значень ваг w_i, що описують кожен вхід нейрона;

- вид активаційної функції f даного нейрона;

- наявність зсуву або порогу активації u (який може бути як додатним, так і від’ємним).

Зважаючи на те, що звичайна ШНМ складається як мінімум із кількох нейронів (а частіше – з десятків, чи навіть сотень нейронів), а в кожного з них є багато входів, то задача вибору вагових коефіцієнтів w_i стає дуже серйозною за своєю складністю. Власне, тривалий процес завдання конкретних «правильних» значень цих вагових коефіцієнтів називають процесом навчання нейронної мережі. Існують різні алгоритми навчання

нейронних мереж, які укрупнено можна поділити на дві групи: «з учителем» та «без учителя». В цілому, навчання нейронних мереж є настільки складним процесом, що має досліджуватися в окремому великому блоці інформації і не може бути розглянуте в рамках даної магістерської роботи.

Вид активаційної функції *f*, очевидно, також є дуже важливим моментом для функціонування нейронів, хоча й у значно меншій мірі, ніж вибір синаптичних вагових коефіцієнтів. На рис. 1.2 наведено найбільш поширені види функцій активації, що найчастіше використовуються на практиці при роботі з нейронними мережами. З точки зору обчислювальної складності більш кращим є використання порогових функцій, що фактично являють собою константну залежність (або хоча б лінійних, але не сигмоїдних).

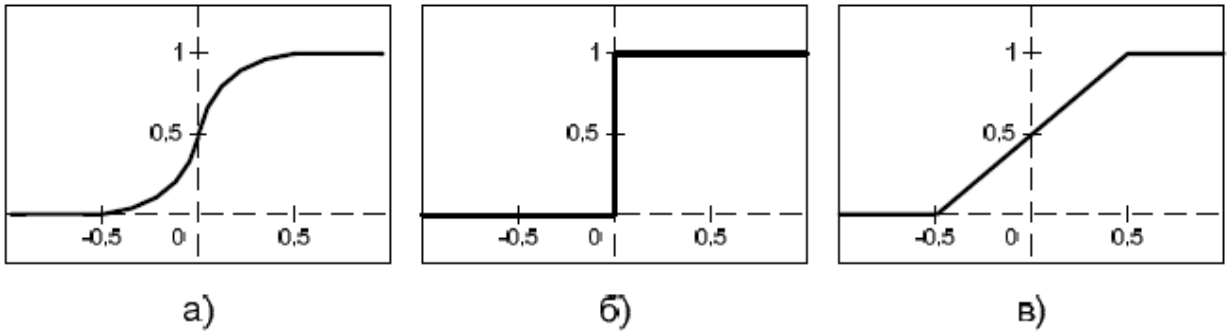

Рис. 1.2. Найбільш поширені типи функцій активації: а – сигмоїдна, б – порогова, в - лінійна.

При побудові цілої нейронної мережі, крім характеристик окремих нейронів, важливим стає також спосіб їх з’єднання (так само, як і їх загальна кількість, організація нейронів у групи, шари, і т.д.).

Найпростішим і в той же час найпоширенішим способом з’єднання нейронів у конкретну ШНМ є персептрон (перцептрон), що обумовлено широким спектром задач, які можна вирішувати за допомогою ШНМ з такою внутрішньою структурою.

Беручи до уваги усі особливості ШНМ, а також усіх сформованих вище кроків по вирішенню задачі прогнозування, бачимо, що з формальної точки зору нейронні мережі є ідеальним інструментом для вирішення задачі прогнозування. В реальності ж ситуація є дещо гіршою: через надзвичайну складність перетворення вхідного сигналу у вихідний, а також через наближений (а не точний) характер результату цього процесу ніхто не може гарантувати, що на досить близьких наборах вхідних параметрів мережі не даватиме кардинально різних відповідей (проблема неперервності та поступової, плавної зміни функції (1.10), яку кінцево і реалізує штучна нейронна мережа у багатовимірному просторі станів системи).

Таким чином, використання нейронної мережі теоретично може мати нестабільний характер і давати погані результати на окремих наборах вхідних даних, навіть при дуже великих обсягах навчальної вибірки та якісно реалізованому процесі навчання. Саме тому часто перевагу надають іншим методам з галузі штучного інтелекту.

Одним із таких популярних методів отримання кінцевого рішення в задачі прогнозування є застосування апарату нечіткої логіки (fuzzy logic), що є ще одним перспективним механізмом, який також широко застосовується в галузі інтелектуального аналізу даних (ІАД) [11-15]. Перевагою таких систем є можливість роботи із нечисловими, розмитими, або взагалі якісними, вираженими словами, лінгвістичними даними. Процедура фазифікації вхідної інформації, в якій використовуються лінгвістичні змінні дозволяє поставити у відповідність елементам множини $\vec{u}_{n+1}$ їх текстові описи, а потім використати простий та зрозумілий набір правил нечітких продукцій для вироблення кінцевого рішення у нечіткій, розмитій формі. Після застосування процедури дефазифікації отримуємо конкретне значення прогнозованої величини. Головною перевагою при такому підході є відсутність необхідності зведення складної математичної моделі (можливо на основі методів багатовимірного регресійного аналізу, інших статистичних методів), а використання простої бази правил, яку може розробити на основі

аналізу логіки роботи системи (або ходу протікання процесу) будь-яка тверезо мисляча людина. Правила мають структуру на зразок «якщо кількість травм серед захисників є високою, то висока імовірність пропустити гол», «якщо нападаючі у гарній формі, то висока імовірність забити гол», і т.п.

Зважаючи на широку поширеність систем нечіткого виводу, розглянемо докладніше весь процес їх використання (зокрема, для задач прогнозування, як у даній роботі). Він полягає у циклічному виконанні процедури нечіткого виводу (див. нижче) за допомогою системи нечіткого виводу. Єдиним кроком, що не виконується під час робочого часу системи, а реалізується лише один раз – при розробці системи – є створення системи нечітких правил продукцій (бази правил нечітких продукцій).

Під системою нечітких правил продукцій мається на увазі узгоджена множина окремих правил нечітких продукцій, побудованих на основі виразів виду:

$$\text{ПРАВИЛО <1>: ЯКЩО «}\beta_1 \text{ є } \alpha_{11}\text{» ТО «}\beta_2 \text{ є } \alpha_{21}\text{»,} \qquad (1.14)$$

де «є» означає «являється», або англійською «IS».

Тут нечіткі вирази антецеденту *A* та консеквенту *B* розписані у формі «β_i є α_{ij}» через лінгвістичні змінні β_i, деякі з яких є вхідними (це змінні, що беруть участь у антецедентах), деякі – вихідними (змінні, які входять до консеквентів). α_{ij} – який-небудь конкретний терм із терм-множини лінгвістичної змінної β_i.

Таким чином, першим кроком процедури нечіткого виводу, який до того ж виконується лише один раз, є створення бази правил нечітких продукцій. Це задача, яку має виконувати експерт (експерти) у даній предметній галузі, оскільки від якості цієї системи (її повноти та адекватності кожного правила) залежить успішна робота системи прогнозування в майбутньому. Помилки при проектуванні бази правил можуть приводити до катастрофічних наслідків, причому найбільша частка складних проблем має за причину саме недоліки системи нечітких правил продукцій.

Важливою властивістю бази правил має бути її узгодженість, тобто правила не повинні суперечити одне одному (так завжди і є, коли розробку виконує один експерт, що має закінчений, усталений і ефективний погляд на прогнозування задач заданого класу). При розробці бази правил командою експертів, перевірка на узгодженість має виконуватися окремим підетапом.

Після зведення бази правил нечітких продукцій та її реалізації у певному електронному апаратному забезпеченні розпочинається процес нормального функціонування системи прогнозування, що є ітеративним, циклічним і кожна ітерація складається з наступних складових (рис. 1.3):

- фазифікація вхідних величин, що входять до умов правил нечітких продукцій;

- агрегування підумов для правил, що мають складені антецеденти;

- активізація підвисновків;

- акумулювання висновків (для тих вихідних змінних, різні терми яких входять до різних активних правил нечітких продукцій, а зазвичай для кожної вихідної змінної так і є);

- дефазифікація вихідних змінних.

Розглянемо докладніше кожен із п’яти повторюваних етапів процедури нечіткого виведення.

а) Першим із них є фазифікація, суть якої полягає у обчисленні значень функцій належності усіх активних термів усіх вхідних лінгвістичних змінних. Даний процес виконується на основі звичайних (не нечітких) даних і має на меті їх переведення у терміни теорії нечітких множин, саме тому фазифікацію ще називають введенням нечіткості.

Ціллю фазифікації є встановлення відповідності між конкретним (зазвичай, числовим) значенням окремої вхідної змінної системи нечіткого виводу та значенням функції належності відповідного їй терма вхідної лінгвістичної змінної. Після завершення цього етапу для всіх вхідних змінних мають бути визначені конкретні значення функцій належності по

кожному із лінгвістичних термів, що використовуються у підумовах бази правил системи нечіткого виводу.

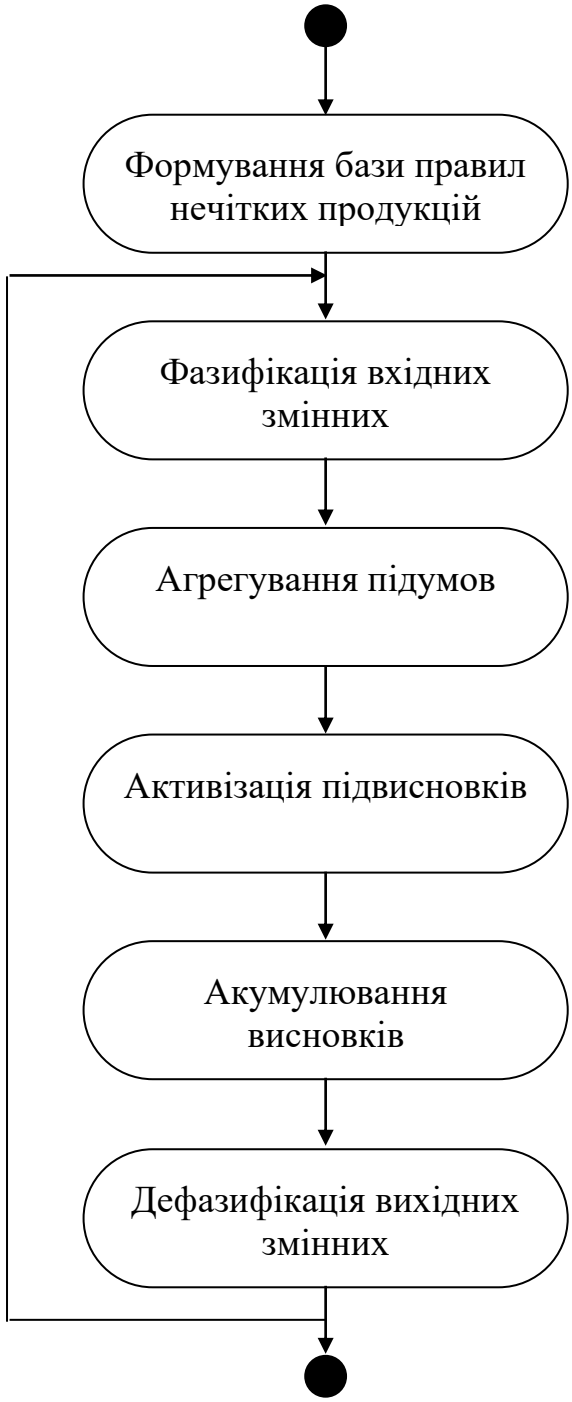

Рис. 1.3. Діаграма діяльності процесу нечіткого виводу (у формі діаграми діяльності мови UML).

б) Наступним етапом процедури нечіткого виводу є агрегування. Воно застосовується до тих правил, які мають складені умови, що складаються з

декількох підумов. Для кожної підумови шляхом виконання процесу фазифікації встановлюється ступінь істинності. По цим даним для кожного правила встановлюється ступінь його істинності в цілому, що виконується на основі об’єднання ступеней істинності окремих підумов, що входять до даного правила (якщо якесь правило побудовано на основі елементарної умови, то ступінь істинності всього правила приймається рівним ступеню істинності цієї єдиної умови). Об’єднання підумов одного правила здійснюється за допомогою операції нечіткої кон’юнкції ТА чи нечіткої диз’юнкції АБО.

Результатом етапу агрегування є встановлення ступеня істинності (у вигляді числа в діапазоні від 0 до 1 включно) для кожного із правил, що утворюють базу нечітких продукцій

в) Активізація є третім із повторюваних етапів процедури нечіткого виводу і полягає у встановленні ступеня істинності кожного із підвисновків усіх правил, що були активними на цій ітерації процесу (тобто таких правил, які на попередньому кроці агрегування мали ненульові результуючі ступені істинності). Ціллю активізації є вироблення ступенів істинності для кожного підвисновку, в результаті чого на наступному кроці можна буде перейти до визначення цілих лінгвістичних змінних, що є вихідними.

г) Акумуляція є передостаннім етапом процедури нечіткого виводу і необхідна через той факт, що різні терми однієї і тієї ж лінгвістичної змінної можуть входити до підвисновків різних правил нечітких продукцій, і тому мають бути якимсь чином об’єднані для формування значення цієї конкретної лінгвістичної змінної.

Результатом акумуляції є сформований набір функцій належності для кожної лінгвістичної змінної, які враховують і відповідають усім термам, задіяним у активованих правилах бази правил нечітких продукцій.

д) Останній етап процедури нечіткого виводу називається дефазифікацією, а полягає він у тому, щоби на основі розмитої інформації,

якою є отримані на попередньому етапі види функцій належності усіх вихідних змінних, обчислити конкретні значення цих змінних, яких їм треба надати.

Для виконання дефазифікації існує багато різноманітних методів, але найбільш популярними є метод центру тяжіння (який також називають просто словом «центроїд») та центру площі, відповідно до яких треба досліджувати геометричну фігуру, обмежену прямими $x = 0$, $x = x_{max}$, $\mu = 0$, $\mu = 1$, а також – лінією, що зображує функцію належності даної вихідної лінгвістичної змінної. У першому випадку слід знаходити координату x центра тяжіння цієї двовимірної фігури, а у другому – координату x вертикальної прямої, що ділить площу цієї фігури на 2 рівних по площі частини.

Після виконання етапу дефазифікації для кожної вихідної змінної буде наявним конкретне числове значення, якому вона повинна дорівнювати для того, щоби забезпечити найкращий результат прогнозування.

Нечіткі системи також мають недолік складності контролювання традиційними математичними методами, а результат прогнозування сильно залежить від вибору функцій належності (який, взагалі кажучи, є вільним). Через ці причини нечіткі множини у даній роботі застосовувати не будемо.

Нарешті, в деяких роботах реалізуються спроби застосувати для прогнозування такі алгоритми та підходи, що, через малу розповсюдженість у дослідженнях, в цілому можна назвати екзотичними. Сюди наприклад, відносять генетичні алгоритми [16], марківські моделі [17], методи поширення очікування [18], хмарні технології [19], і т.п.

1.3. Уточнена постановка задачі дослідження

Розглянувши у попередньому підрозділі існуючі методи прогнозування результатів спортивних подій, можна зробити висновок, що:

- серед науковців по даний час продовжуються активні пошуки ефективних способів прогнозування в окресленій галузі;

- задача удосконалення існуючих методів прогнозування є актуальною та потребує нагального вирішення;

- серед наведених вище проаналізованих способів прогнозування існують різноманітні можливості підвищення їх ефективності різними шляхами та засобами.

Відповідно, можна сформулювати уточнену постановку задачу дослідження: на основі існуючих методів прогнозування результатів спортивних подій вибрати метод, що підлягає удосконаленню, та здійснити покращення цього алгоритму, реалізувати це покращення програмно у висхідних кодах та протестувати його на конкретних вхідних даних.

Висновки по розділу 1

Таким чином, у даному розділі виконано дослідження сучасних методів прогнозування, зокрема у галузі спортивних змагань. Встановлено, що на сьогоднішній день активно розробляються:

- методи математичної статистики;

- підходи по застосуванню нейронних мереж;

- методи нечіткої логіки;

- деякі екзотичні методи, що пропонуються із інших галузей окремими дослідниками.

В цілому, встановлено, що в галузі застосування статистичних методів існують можливості по їх удосконаленню, як, в цілому, і для інших підходів, чому і буде присвячено наступні дослідження.

РОЗДІЛ 2. РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДЛЯ ПІДВИЩЕННЯ ЯКОСТІ ПРОГНОЗУВАННЯ СПОРТИВНИХ ПОДІЙ

2.1. Аналіз та пошук можливостей удосконалення існуючих методик прогнозування

Як було докладно розібрано у попередньому розділі, існують розроблені математичні методи, що дозволяють здійснювати більш-менш точне прогнозування результатів спортивних подій, робота яких, зокрема, розбирається на прикладі прогнозу результату футбольних матчів (хоча, за необхідності, може бути легко перенесена на інші види спорту при повторенні та адаптації усієї процедури для відповідного напрямку). Проаналізуємо, які з цих методів підлягають модифікації для підвищення показників прогнозованості.

Методи на основі залучення нейронної мережі використовують готовий об’єкт – безпосередньо ШНМ – який початково не було створено виключно для вирішення задач прогнозування. Початково була сформована сама концепція штучних нейронів та їх композицій (мереж), а вже після цього (зважаючи на красиву простоту, універсальність та ефективність цієї концепції) почалися масштабні пошуки її галузей застосування. Таким чином, вносити зміни «зверху-вниз», тобто від потреб галузі застосування до фундаментальної концепції, що лежить в основі всього підходу у даному випадку є не зовсім логічно. Дійсно, внесення таких змін означало би розробку якогось нового типу нейронних мереж в цілому, або внесення суттєвих покращень у їх роботу взагалі (а не тільки для прогнозування), що відноситься до зовсім іншої обширної наукової галузі і не може розглядатися в рамках даного дослідження. Таким чином, через глобальність (складність) концепції штучних нейронних мереж, їхню певну інкапсульованість (закритість), вносити якісь удосконалення в рамках даної роботи не уявляється можливим.

Схожа ситуація спостерігається і з застосуванням методів теорії нечітких множин (ТНМ). Так, існує цілком конкретна процедура нечіткого виведення, зафіксована в самих основах ТНМ (показана на рис. 1.3), вносити зміни до якої, роблячи якісь удосконалення, не уявляється можливим. Звичайно, окремі етапи цієї процедури можуть виконуватися різними шляхами, наприклад, лише для етапу дефазифікації запропоновано більше десятка різних способів проведення. Аналогічною є ситуація і з іншими етапами, кожен з яких може виконуватися багатьма способами, тому при комбінаторному урахуванні всіх можливих варіантів проведення кожного з п’яти етапів отримуємо загальну кількість конкретних способів реалізації процедури нечіткого виведення порядку тисяч (наприклад, якщо кожен з п’яти етапів може виконуватися хоча б трьома способами, то загальна кількість різних варіантів уже рівна $3^5 = 243$, але, як уже було сказано вище, число способів проведення кожного етапу значно більше, ніж 3).

Однак, велика кількість варіантів проведення процедури нечіткого виведення є плюсом лише для довготривалих або масштабних оплачуваних досліджень, у яких можуть бути задіяні цілі колективи науковців. В рамках виконання магістерської роботи ця особливість навпаки є недоліком, оскільки в умовах стиснутих часових рамок та необхідності отримати конкретний результат обмеженими силами (однією особою, а не колективом), навряд чи можна досягти мети, дослідивши кілька варіантів (або, навіть і десятків варіантів) із тисяч і тисяч можливих.

Таким чином, удосконалення апарату нечіткої логіки з метою покращення показників прогнозування результатів спортивних подій також не уявляється можливим, в першу чергу, через величезну кількість доступних шляхів такого покращення та неможливість заранні встановити перспективні з них без простого повного перебору усіх доступних варіантів.

Таким чином, можна перейти до аналізу більш традиційних, математично вивірених методів прогнозування. Так у роботі [1]

пропонується підхід на основі кваліметрії, який і буде прийнято за основний, тому розглянемо його докладніше.

Кваліметрія - це наука про методи кількісного оцінювання якості. Сутність кваліметричного підходу полягає в тому, що будь-яке якісне явище можна розкласти на фактори і критерії, які допомагають краще розкрити це явище. В нашому випадку критеріями будуть виступати складові вектору $\vec{u}_i$.

Як і у будь-якій багатокритеріальній задачі, кожній складовій u_{ij} доцільно присвоїти якусь вагу w_j, оскільки очевидно, що деякі показники мають більший вплив на кінцеве рішення $\vec{v}$, ніж інші, які менш важливі, але теж потребують урахування.

В інших задачах прогноз може мати складний характер $\vec{v}_i$, однак в задачі прогнозування саме результату спортивних подій, зазвичай, гарно було би отримати одне (але якомога надійніше) значення, що надавало би інформацію про переможця поєдинку. Таким чином, у галузі, що розглядається, достатньо всього лише однієї координати v_i:

$$\vec{v}_i = \{v_{i1}\} = \{v_i\}. \qquad (2.1)$$

Беручи до уваги (2.1), можна тоді прогнозом (у вузькому сенсі) називати не вектор $\vec{v}_{n+1}$, а єдину його складову (тобто скалярну величину) v_{n+1}.

При таких допущеннях прогнозоване значення може розраховуватися за різними формулами, з яких розглянемо, наприклад, три наступних:

1) мультиплікативна формула:

$$v_{n+1} = \frac{\prod_{r=1}^{m_{u1}} \left(u_{(n+1)r} \cdot \rho_{(n+1)r} \cdot w_{(n+1)r} \right)}{\prod_{r=m_{u1}+1}^{m_u} \left(u_{(n+1)r} \cdot \rho_{(n+1)r} \cdot w_{(n+1)r}^{-1} \right)},$$

де m_{u1} – кількість вхідних показників, при збільшенні яких підвищуються шанси на перемогу, такі показники, відповідно до термінології [1], будемо називати стимулюючими (показники-стимулятори);

$\rho_{(n+1)r}$ – коефіцієнт для приведення до єдиної міри (розмірності) r-того показника, нормалізуючий множник, який часто розраховується як $1/u_{ir\max}$.

При такому підході також слід розглядати константу m_{u2} – кількість вхідних показників, при зменшенні яких підвищуються шанси на перемогу (їх будемо називати дестимулюючими, показники-дестимулятори). Справедливим є співвідношення:

$$m_{u1} + m_{u2} = m_u.$$

2) формула на базі теорії адитивної корисності:

$$v_{n+1} = \frac{\sum_{r=1}^{m_{u1}} \left(u_{(n+1)r} \cdot \rho_{(n+1)r} \cdot w_{(n+1)r} \right)}{\sum_{r=m_{u1}+1}^{m_u} \left(u_{(n+1)r} \cdot \rho_{(n+1)r} \cdot w_{(n+1)r}^{-1} \right)}$$

3) формула згідно теорії адаптивної корисності:

$$v_{n+1} = \sum_{r=1}^{m_{u1}} \left(u_{(n+1)r} \cdot \rho_{(n+1)r} \cdot w_{(n+1)r} \right) + \sum_{r=m_{u1}+1}^{m_u} \left(\frac{w_{(n+1)r}}{u_{(n+1)r} \cdot \rho_{(n+1)r}} \right). \qquad (2.2)$$

Останній варіант і узято в [1] та ряді схожих робіт (з невеликими варіаціями) за основу.

Конкретніше, методика прогнозування переможця футбольного матчу з [1] складається з трьох кроків:

1) Обчислення відносних значень кожного показника (тобто приведення до єдиної міри), іншими словами це аналог обчислення добутків $u_{(n+1)r} \cdot \rho_{(n+1)r}$;

2) Обчислення рейтингу обох команд шляхом множення кожного нормалізованого показника на його ваговий коефіцієнт та сумування (іншими словами це обчислення згортки виду (2.2);

3) Нормування рейтингу обох команд на 1 (необов’язковий етап, який виконується для зручності порівняння двох показників), що виконується по формулам:

$$v_{rel} = \frac{v_{n+1}}{v_{n+1} + v_{n+1}'}; \quad v_{rel}' = \frac{v_{n+1}'}{v_{n+1} + v_{n+1}'}, \qquad (2.3)$$

де v_{n+1} – прогнозоване значення рейтингу для першої команди;

v_{rel} – відносне значення рейтингу першої команди (основне значення прогнозу).

v_{n+1}' – значення рейтингу для другої команди;

v_{rel}' – відносне значення рейтингу другої команди.

У формулі (2.3) і далі по тексту записки штрихами позначено усі величини, що відносяться до другої команди.

Кінцево, із двох отриманих відносних показників (що є доповненням один одного до одиниці) береться один v_{rel} і по ньому робиться прогноз за лінгвістичною шкалою, наведеною в табл. 2.1 (яка також пропонується в [1]).

Табл. 2.1. Лінгвістична шкала для визначення результату матчу за відносним рейтингом команд.

Результат першої команди	Рейтинг команди
Перемога	$v_{rel} > 75\%$
Перемога або нічия	$55\% < v_{rel} <= 75\%$
Нічия	$45\% <= v_{rel} < 55\%$
Поразка або нічия	$25\% <= v_{rel} < 45\%$
Поразка	$v_{rel} < 25\%$

При розробці власних лінгвістичних шкал слід брати до уваги те, що вони мають бути дзеркально симетричними (наприклад, якщо для перемоги команди приймається $v_{rel} > 80\%$, умовою поразки має бути $v_{rel} < 20\%$, і т.д.).

На основі простих перетворень можна отримати кінцеву точну формулу, що відповідає крокам 1-2:

$$v_{n+1} = \sum_{r=1}^{m_{u1}} \left(u_{(n+1)r} \cdot \frac{1}{u_{(n+1)r} + u_{(n+1)r}'} \cdot w_{(n+1)r} \right) +$$

$$+ \sum_{r=m_{u1}+1}^{m_u} \left(\left(1 - \frac{u_{(n+1)r}}{u_{(n+1)r} + u_{(n+1)r}'} \right) \cdot w_{(n+1)r} \right) =$$

$$= \sum_{r=1}^{m_{u1}} \left(\frac{u_{(n+1)r}}{u_{(n+1)r} + u_{(n+1)r}'} \cdot w_{(n+1)r} \right) + \sum_{r=m_{u1}+1}^{m_u} \left(\frac{u_{(n+1)r}'}{u_{(n+1)r} + u_{(n+1)r}'} \cdot w_{(n+1)r} \right) \qquad (2.4)$$

Відмітимо, що у якості окремих показників u_{ij} для задачі прогнозування результату саме футбольних матчів доцільно брати наступні (через тире наводяться вагові коефіцієнти, рекомендовані в [1], і які беруться за десятибальною шкалою):

1. Кількість пропущених м’ячів – 7.

2. Поспіль матчів, в яких пропускали голи – 5.

3. Набрано очок – 9.

4. Кількість забитих м’ячів – 7.

5. Кількість ударів по воротах – 4.

6. Кількість ударів в створ воріт – 7.

7. Кількість кутових – 3.

8. Відсоток володіння м’ячем – 3.

9. Днів відпочинку – 4.

10. Поспіль матчів, в яких забивали голи – 5.

11. Домашній матч – 7.

12. Місце команди в турнірній таблиці – 6.

Коефіцієнти 1, 2 – дестимулюючі (чим менше показник, тим більше буде рейтинг), а всі інші – стимулюючі (чим більше показник, тим більше буде рейтинг). Показник 11 рівний 1, якщо матч домашній, 0 – у протилежному випадку.

Даний метод описаний у різних джерелах із невеликим змінами, і при цьому декларується його ефективність на рівні 60-90%, отже, навіть у такій формі підхід ефективний, а саме, кращий, ніж прямий вибір «наосліп».

Недоліком описаного методу, що надає добре контрольований результат, є відповідність формулі (1.10), в той час, як залежність (1.7) дозволяє взяти до уваги додаткову (і важливу для випадку, що розглядається!) інформацію **X** про попередні стани процесу, а, отже, може надавати більш точні прогнозні результати.

Дана пропозиція задає напрямок удосконалення методу прогнозування результатів спортивних подій на основі кваліметрії: в нього треба внести урахування показників попередніх матчів. Визначившись із концепцією, можна переходити до конкретних кроків, що висвітлені у наступному підрозділі.

2.2. Розробка алгоритмів удосконалення методів прогнозування результатів спортивних подій

Говорячи конкретніше, для розрахунку прогнозних значень слід враховувати не лише комплекс показників $\vec{u}_{n+1}$, що описують даний матч, який прогнозується, а й такі ж комплекси $\vec{u}_i$, що описували попередній матч, той що передував попередньому, і т.д., всього, згідно (1.1) p попередніх станів. Також мають ураховуватися результати цих попередніх матчів $\vec{v}_i$, одним словом, згідно (1.4) урахуванню підлягають вектори $\vec{x}_i$, починаючи з $\vec{x}_n$ і закінчуючи $\vec{x}_k$, всього p різних станів. Така інформація повністю відповідає матриці **X**, що була запропонована в (1.7), і яка має ураховуватися при виведенні підсумкового рейтингу команд.

Таке урахування пропонується окремо виконувати для складових $\vec{u}_i$ цієї матриці та для результатів попередніх матчів $\vec{v}_i$.

Комплекси вхідних змінних $\vec{u}_i$ для попередніх матчів можуть ураховуватися по залежності, аналогічній (2.4), але у неї слід ввести певний коефіцієнт застарівання q_i, що зменшував би вплив на підсумковий результат тієї інформації, що отримана давно. Загальний рейтинг тоді можна знаходити як зважену суму складових виду (2.4):

$$v_{n+1} = \sum_{i=k}^{n+1}\left(q_i \left(\sum_{r=1}^{m_{u1}} \left(\frac{u_{ir}}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) + \sum_{r=m_{u1}+1}^{m_u} \left(\frac{u_{ir}'}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) \right) \right). \qquad (2.5)$$

Коефіцієнт застарівання q_i можна лінійно зменшувати від 1 для $i = n + 1$ до 0 для $i = k - 1$:

$$q_i = \frac{i - k + 1}{n - k + 2}. \qquad (2.6)$$

Приклад поведінки коефіцієнту застарівання показано на скріншоті з середовища Mathcad, що використовувалося у даній роботі у якості допоміжного засобу – рис. 2.1.

$k := 3 \quad n := 7 \quad i := k..n + 1$

$$q_i := \frac{i - k + 1}{n - k + 2}$$

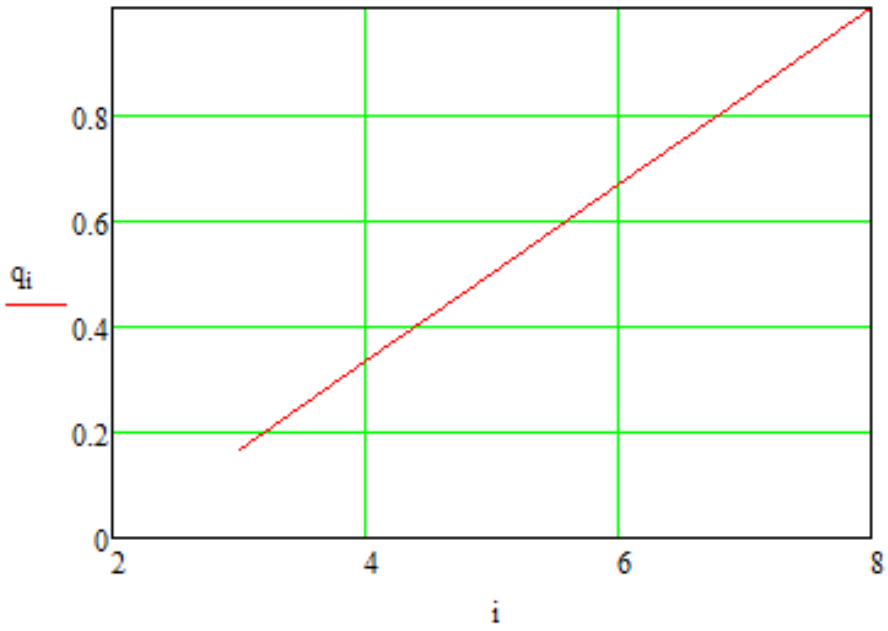

Рис. 2.1. Приклад поведінки коефіцієнту застарівання для різних *i*.

Оскільки головним у формулах (2.5)-(2.6) є номер n (а саме, номер прогнозованого стану позначено через $n + 1$), то не зовсім зручно кожного разу вручну вираховувати номер k, з якого починаються стани, що ще ураховуються при прогнозуванні. Краще перейти до використання кількості таких станів, що ураховуються, яку в 1 розділі було позначено p:

$$k = n + 1 - p \qquad (2.7)$$

З урахуванням (2.7) вирази (2.5)-(2.6) набувають вигляду:

$$v_{n+1} = \sum_{i=n+1-p}^{n+1} \left(q_i \left(\sum_{r=1}^{m_{u1}} \left(\frac{u_{ir}}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) + \sum_{r=m_{u1}+1}^{m_u} \left(\frac{u_{ir}'}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) \right) \right). \quad (2.8)$$

$$q_i = \frac{i - n - 1 + p + 1}{n - n - 1 + p + 2} = \frac{i - n + p}{p + 1}. \quad (2.9)$$

Підставляючи (2.9) у (2.8) отримуємо формулу для прогнозованого значення v_{n+1}, що враховує інформацію про попередні стани учасника змагань:

$$v_{n+1} = \sum_{i=n+1-p}^{n+1} \left(\frac{i - n + p}{p + 1} \cdot \left(\sum_{r=1}^{m_{u1}} \left(\frac{u_{ir}}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) + \sum_{r=m_{u1}+1}^{m_u} \left(\frac{u_{ir}'}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) \right) \right) \quad (2.10)$$

У (2.10) не враховано результати останніх p матчів, що також мають входити у кінцеву формулу з коефіцієнтами застарівання q_i. Зважаючи на те, що відомості про перемоги, чи програші є чи не найбільш важливою для прогнозування інформацією, пропонується включити її у стимулюючі показники із найбільшим ваговим коефіцієнтом (як зазначалося вище, це 10). Крім того, не є відомими значення рейтингу команд v_i та v_i’ у попередніх іграх, а лише відомо результат гри (перемога, нічия або програш), отже для урахування цієї інформації у кінцевому рішенні слід ввести додаткові позначення для цих трьох випадків. За умови виграшу попередньої гри приймаємо $V_i = 1$, програшу $V_i = 0$, а якщо була нічия $V_i = 0{,}5$. Аналогічно і для другої команди знаходимо V_i’.

В результаті маємо кінцеву формулу для розрахунку рейтингу команди перед матчем, яка враховує і поточний набір показників команди, і її передісторію (причому із відповідними коефіцієнтами застарівання інформації):

$$v_{n+1} = \sum_{i=n+1-p}^{n} \left(\frac{i - n + p}{p + 1} \cdot \left(\sum_{r=1}^{m_{u1}} \left(\frac{u_{ir}}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) + 10 \frac{V_i}{V_i + V_i'} + \sum_{r=m_{u1}+1}^{m_u} \left(\frac{u_{ir}'}{u_{ir} + u_{ir}'} \cdot w_{ir} \right) \right) \right) +$$

$$+ \left(\sum_{r=1}^{m_{u1}} \left(\frac{u_{(n+1)r}}{u_{(n+1)r} + u_{(n+1)r}'} \cdot w_{(n+1)r} \right) + \sum_{r=m_{u1}+1}^{m_u} \left(\frac{u_{(n+1)r}'}{u_{(n+1)r} + u_{(n+1)r}'} \cdot w_{(n+1)r} \right) \right) \quad (2.11)$$

Винесення останнього доданку (при $i = n + 1$) в (2.11) окремо від загальної суми обумовлено тим, що в ньому має бути відсутнім результат прогнозування V_i (адже він власне і підлягає визначенню).

За формулою (2.11) вираховуються рейтингові показники (v_{n+1} та v_{n+1}') для обох команд, що беруть участь у матчі, а потім, як і в [1] використовується нормуюча формула (2.3) та лінгвістична шкала з табл. 2.1.

Таким чином, удосконалено широко відомий алгоритм з [1] і отримано нову формулу для розрахунку підсумкового рейтингу команди у вигляді (2.11).

Висновки по розділу 1.

Таким чином, у розділі докладно розглянуто особливості популярних методів прогнозування, причому саме у їх застосуванні до задачі передбачення результатів спортивних подій (у тих місцях, де необхідною є повна конкретика, мова йшла про такий поширений і дуже популярний вид спорту, як футбол; під подією тоді мався на увазі один футбольний матч). Встановлено, що такі популярні методи з галузі штучного інтелекту, як прогнозування на основі нейронних мереж та нечіткої логіки не можуть бути ефективно застосовані в рамках даного дослідження через організаційні складнощі (удосконалення в галузі нейронних мереж потребує фундаментальних змін у самій структурі або методах роботи з нейронними мережами, що є набагато більш широкою, ніж прогнозування, і окремою важливою задачею галузі ІТ; в галузі нечіткої логіки малоймовірно в умовах обмеженого часу та ресурсів натрапити на конкретний новий хід процедури нечіткого виведення, що даватиме кращі результати саме у прогнозуванні, зважаючи на тисячі можливих альтернативних варіантів проведення такої процедури). Більш контрольованими та пристосованими для модифікації в рамках магістерського дослідження є статистичні кваліметричні методи.

На основі відомого методу, який є досить поширеним, запропоновано удосконалення, що полягає в урахуванні не лише поточних показників, що

відповідають прогнозованому матчеві, а й попередніх періодів (причому із пропорційно зменшуваними коефіцієнтами застарівання інформації). Результати попередніх матчів ураховуються на основі штучної шкали (0 – поразка, 0,5 – нічия, 1 – перемога).

В цілому даний підхід є новим і створений саме в рамках магістерського дослідження. В наступних розділах розроблені формули буде реалізовано програмно та досліджено на ефективність.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДЛЯ ПРОГНОЗУВАННЯ СПОРТИВНИХ ПОДІЙ

3.1. Обґрунтування вибору технології програмування

Першочерговим питання, яке постає перед розробниками практично будь-якого програмного забезпечення, є вибір моделі або технології його розробки. Такими, що реально використовуються на сьогоднішні день у виробничій практиці, є технології структурного (процедурного) та об’єктно-орієнтованого програмування. Кожна з них має свої особливості, переваги і недоліки, які розглянемо докладніше.

Структурне, або як його ще називають практикуючі програмісти, процедурне програмування засноване на використанні окремих структурних блоків - в першу чергу, підпрограм (процедур і функцій).

Історично перші комп’ютерні програми були відносно простими і мали пакетний режим роботи: отримуючи на вхід якусь інформацію (можливо, навіть на перфокарті) вони виконували певний обсяг операцій з обробки цих даних і видавали результат. При цьому існувала сувора функціональна залежність виходу від входу – рис. 3.1.

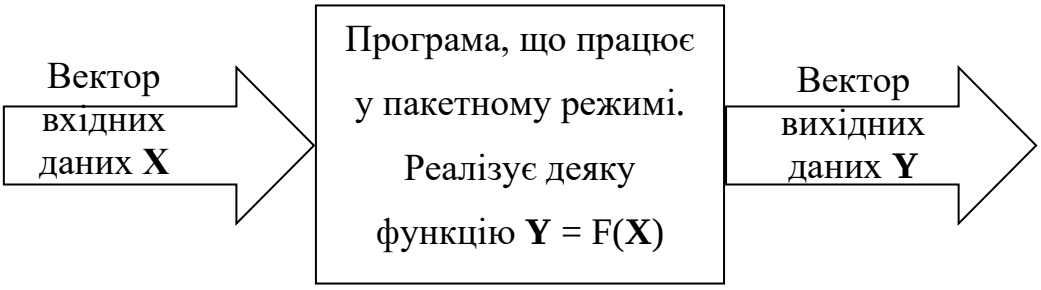

Рис. 3.1. Схема роботи пакетної програми.

Незважаючи на відсутність явного зв’язку функціональності і структури, пакетні програми в основному мали просту лінійну послідовність виконання. Тобто в них практично відсутні будь-які підпрограми, принаймні, використання підпрограм не було наріжним каменем самої методики програмування.

В міру ускладнення функціональності програмного забезпечення, змінювалася і його внутрішня структура: поступово розвинулася інтерактивна модель взаємодії користувача і програми – рис. 3.2. Програми стали запитувати інформацію і активно реагувати на дії людини. Ускладнення функціональності привело до відповідного ускладнення програмного коду, якого, в першу чергу, стало просто багато. Багато для того, щоб людина-програміст без всяких спеціальних хитрувань швидко і легко розібралася з незнайомим кодом. Розміщувати код у простій лінійній послідовності без виділення великих блоків стало незручно, в першу чергу, для розуміння цього коду.

Рис. 3.2. Схема роботи програми, що активно взаємодіє з користувачем.

Тут слід зазначити психологічні особливості сприйняття людиною складних «великих» завдань. Неструктуроване «велике» завдання (наприклад, написання дипломної роботи) зазвичай викликає певний психологічний ступор і, як результат, повну неможливість поступово розібратися з ним. Людині зручно розбити проблему на не надто велику (зазвичай до десятка, а краще 3-4) кількість завдань (наприклад, розділів у дипломній роботі), не замислюючись про реалізацію кожного з них. Коли є ясність і розуміння проблеми на найвищому рівні абстракції, слід приступати до деталізації підзадач, кожну з яких слід розбити на окремі «підпідзадачі» тобто підзадачі нижчого рівня, більш дрібні. Уже після такого розбиття слід аналізувати всі перераховані підзадачі. На певному етапі зупиняються і

виконують не розбиття чергової підзадачі на більш дрібні, а безпосередню її реалізацію в програмних кодах.

Отже, можна сказати, що розвиток методики програмування відбувався в ногу з розвитком призначеного для користувача інтерфейсу: і грубо кажучи, розвиненому інтерфейсу командного рядка відповідає парадигма структурного програмування.

Говорячи більш строго, структурне програмування має на увазі побудова програми відповідно до трьох основних принципів: слідування, розгалуження, повторення.

Слідування має на увазі, що оператори і блоки програмного коду слідують і виконуються один за іншим. Розгалуження реалізується різними умовними операторами типу if, і дозволяє вибирати один з декількох подальших варіантів виконання програми. Повторення зазвичай відносять на рахунок циклів (що йдуть підряд багаторазових повторів одного і того ж ділянки коду), хоча цей же принцип можна віднести і до підпрограм.

Взагалі ж, структурне програмування у деякій мірі є застарілою методикою програмування, на зміну якій разом з віконним інтерфейсом прийшло об’єктно-орієнтоване програмування (деякі сучасні мови програмування загального призначення навіть не дозволяють створити структурну програму, тільки об’єктно-орієнтовану – як, наприклад, Visual C# чи Java). Проте, при створенні невеликих програм (наприклад, до 10000 рядків коду і без передбачуваного розширення) застосування цієї методики програмування більш виправдано і код краще сприймається, ніж його об’єктно-орієнтований варіант.

Суть же методології об’єктно-орієнтованого програмування полягає в тому, що система розглядається, як сукупність окремих сутностей - об’єктів, які мають набір якихось своїх внутрішніх параметрів - властивостей, а також можуть взаємодіяти між собою за допомогою деяких дій - викликів методів (або трохи більше непрямим чином - шляхом надсилання повідомлень, оброблюваних методами об’єктів; для цього необхідна присутність активної

сутності, яка роздає повідомлення адресатам, як, наприклад, менеджер вікон в ОС Windows).

Якщо говорити про програмний код, то для того, щоб оперувати об’єктом, його спочатку потрібно створити. Об’єкти створюються як змінні, у яких типом виступає клас об’єкта. Клас - це просто опис, які властивості можуть мати об’єкти такого типу (тобто яку інформацію вони можуть зберігати), і які у них є методи (тобто які дії вони можуть виконувати). Об’єкт - це набір значень, чому саме рівні властивості даного об’єкта (свої методи кожен об’єкт отримує від свого класу, тобто методи однакові у всіх об’єктів, що належать даному класу).

Для чого потрібен цей специфічний підхід, адже самі по собі об’єкти не додають нічого корисного (навпаки, введення об’єктів ускладнює програму, вносить в неї нові сутності)? Виявляється, реалізовуючи всі сутності, необхідні, згідно з алгоритмом, для роботи програми, у вигляді класів і об’єктів, ми спрощуємо її розуміння для самих себе. Саме тому ОО-підхід рекомендується до застосування для великих проектів (більше десятків тисяч рядків коду), коли утримувати «в голові» всю систему цілком стає важко. Можна сказати, що розбиття програми на об’єкти і проектування їх класів наближає розуміння предметної області до звичного людського образу мислення (в разі «великих» проектів). Людина мислить класами, об’єктами і зв’язками між ними.

Порівняльна складність проектів, що мають однакову функціональність, але побудованих по-різному (згідно структурному і об’єктно-орієнтованого підходів до програмування), як функція їх обсягу показана на рис. 3.3. З графіка рис. 3.3 слід, що, якщо потрібно реалізувати продукт з невеликою функціональністю (тобто кількість рядків коду, що її реалізують буде очевидно невеликою, порядку кількох тисяч рядків), то це краще робити без застосування класів, так як вони будуть тільки ускладнювати всю справу. Якщо ж програма має більш-менш значну функціональність, а значить, реалізується хоча б кількома тисячами рядків

коду, то вже є сенс замислюватися про застосування об’єктно-орієнтованого підходу. Однозначно будуватися за принципами ООП повинні програми, які мають 10000 рядків коду і більш.

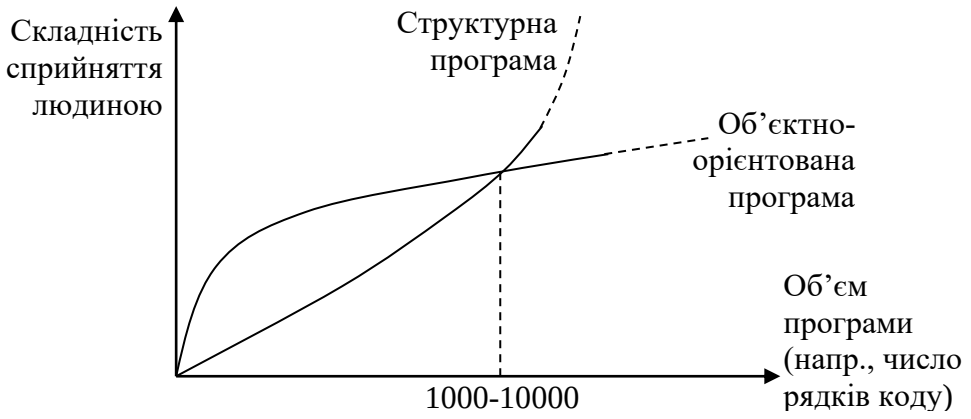

Рис. 3.3. Порівняльна складність висхідного тексту двох програм, що мають однакову функціональність, але реалізованих по-різному: згідно об’єктно-орієнтованому та структурному підходам.

Відзначимо, що часто крім розглянутих міркувань, також на вибір методики програмування впливають інші чинники, наприклад, можливість майбутнього розширення функціональності, створення якомога більш зрозумілого коду (для роботи над проектом цілої команди, а не одного програміста), або просто побажання замовника застосувати найбільш сучасний підхід до програмування.

Крім розбиття (декомпозиції) всієї предметної області на об’єкти (класи) і співвідношення між ними, також ОО-підхід має на увазі дотримання трьох основних його принципів: інкапсуляція, наслідування, поліморфізм.

Під інкапсуляцією мається на увазі об’єднання даних (значення властивостей класу у деякого конкретного об’єкта) та засобів їх обробки (методи класу). Це знову ж таки зручно психологічно, так як дозволяє реалізовувати окремі завершені сутності - класи, які самі обробляють свої дані. Звернення до об’єктів цих класів відбувається за допомогою методів, що утворюють інтерфейс класу.

Наслідування дуже корисно, тому що дозволяє сильно скоротити обсяги повторюваного коду (до чого потрібно завжди прагнути при розробці будь-якого програмного забезпечення). Згідно з цим принципом виділяється клас, який має загальний набір властивостей і методів для декількох більш розширених класів. Цей клас оголошується батьком, базовим класом для декількох похідних від нього (нащадків, спадкоємців). Всі класи-нащадки успадковують від базового всі його властивості та методи, але до цього ще мають свої власні оригінальні властивості і / або методи.

Наприклад, клас Студент є похідним від класу Людина, тому що кожна людина має властивість Ім’я, Прізвище, метод Відпочити(). Однак у Студента є свої специфічні властивості і / або методи, які як раз і відрізняють його від просто Людини: СереднійБал, НомерЗаліковки, ЗдатиЕкзамен(), і т.д.

При наслідуванні іноді методи батьківського і похідного класу мають однакове призначення, але реалізовуються по-різному. Такі методи називаються перевантаженими. Наприклад, метод Відпочити() у класу Людина реалізується як відпочинок на дивані, а у класу Студент - як похід в клуб. При цьому ще раз підкреслимо, що призначення методу в обох випадках одне і той саме.

Поліморфізм є можливістю деякої функції приймати об’єкти як батьківського, так і похідних класів, і вміти викликати перевантажені методи саме того класу, об’єкт якого був переданий в функцію. Слід сказати, що це досить специфічна можливість і в загальному багато програмістів використовують ОО-підхід і без звернення до поліморфізму.

Нехай, наприклад, у програмі є функція ПровестиВихідні(), припустимо яка не належить якомусь класу (хоча це не принципово). Нехай аргументом цієї функції є об’єкт класу Людина. Тоді в неї можна передавати об’єкти всіх похідних від Людини класів: Студент, Службовець, Пенсіонер, і т.д., тому що всі вони є Людиною (спадкоємці цього класу). Ясно, що ця функція повинна включати різні дії: ПрибратиКвартиру(), ПітиНаРинок(), і в

тому числі Відпочити(). Так ось поліморфізм дозволяє всередині цієї функції просто вказати назву методу Відпочити(), не вказуючи якого саме класу він повинен бути викликаний, а вже в процесі виконання програми, якщо в функцію переданий об’єкт класу Студент, то викликається саме його метод Відпочити(), а якщо переданий об’єкт класу Пенсіонер, то автоматично викликається саме його метод Відпочити(), і т.д. Кажуть, що функція ПровестиВихідні() - поліморфна, і вона є такою завдяки тому, що реалізує принцип поліморфізму.

Важливими поняттями в ООП також є: статичні члени класу, абстрактні методи і класи, дружба функцій і класів, і т.д.

Грунтуючись на перерахованих особливостях двох існуючих методів програмування, вибираємо структурний підхід як більш простий, що відповідає невеликим (не промисловим) масштабам проектованого ПЗ, а також вимогам до складності (програма не повинна бути складною через цільову аудиторію користувачів, якими є не спеціалісти з галузі ІТ, а звичайні вболівальники).

3.2. Вибір мови програмування

Після вибору технології програмування наступним кроком є вибір мови програмування (або комплексу мов – за умови, наприклад, написання програмного продукту у вигляді розподіленого клієнт-серверного додатку, а також при деяких інших умовах).

Найбільш укрупнено за типом платформи кінцевого програмного продукту усі мови програмування можна розподілити на засоби для розробки настільних додатків, засоби веб-розробки та мобільної розробки. Мобільні технології в даній роботі взагалі не можуть стати в нагоді, оскільки розроблюваний програмний продукт не потребує мобільності, а обчислювальні ресурси мобільних систем в рази менше, ніж у настільних, чи тим більше, клієнт-серверних. Що ж стосується альтернативи «веб-додаток»

чи «настільне програмне забезпечення», перший варіант видається зручнішим через наступні причини:

- є повністю кросплатформенним (якісні веб-додатки однаково функціонально працюють у будь-яких сучасних веб-браузерах, незалежно від того, на якій операційній системі вони запущені);

- дозволяє використовувати обчислювальні ресурси серверу, які завжди значно потужніші, ніж у настільних ПК;

- можуть мати досить швидкий та зручний інтерфейс, за умови використання у них сучасних веб-технологій (типу AJAX, кешірування, локальної доробки даних, предзавантаження сторінок, і т.д.).

Таким чином, обираємо веб-розробку та проаналізуємо засоби, що можуть при такому випадку бути використані.

Сучасні засоби для розробки Інтернет-ресурсів є настільки обширними, що розібрати їх усі в рамках дипломної роботи абсолютно не уявляється можливим. Тому в процесі вибору будемо орієнтуватися як на об’єктивні фактори (такі, як, наприклад, підтримка сучасних технологій програмування), так і суб’єктивні, але такі, що набули широкого впливу (як-то схильність окремих людей до певних мов, чи, навіть, стилів програмування, що у результаті спричинює популярність відповідних засобів розробки на масовому рівні).

Таким чином, розглянемо найпоширеніші засоби розробки, що є популярними на даний момент у відповідній галузі.

В цілому увесь процес веб-розробки традиційно ділять на дві компоненти: front-end та back-end – рис. 3.4.

Засоби Front-end розробки.

Спрощено кажучи, під «передньою» частиною – Front-end – у програмуванні мають на увазі те, що бачить користувач. Для настільних систем під цим мають на увазі інтерфейс, а під back-end’ом мають на увазі логіку роботи програми, яка теоретично (зокрема, так часто роблять у Linux-системах) взагалі може бути реалізована окремою програмою із текстовим

консольним інтерфейсом. Задачею фронт-енду при цьому є зручний для широкого кола користувачів збір усіх вхідних даних, необхідних для виконання змістовних операцій консольною утилітою back-end'ом. У багатьох же випадках фронт та бек поєднані разом в рамках одного програмного забезпечення. Такою є ситуація для настільних систем.

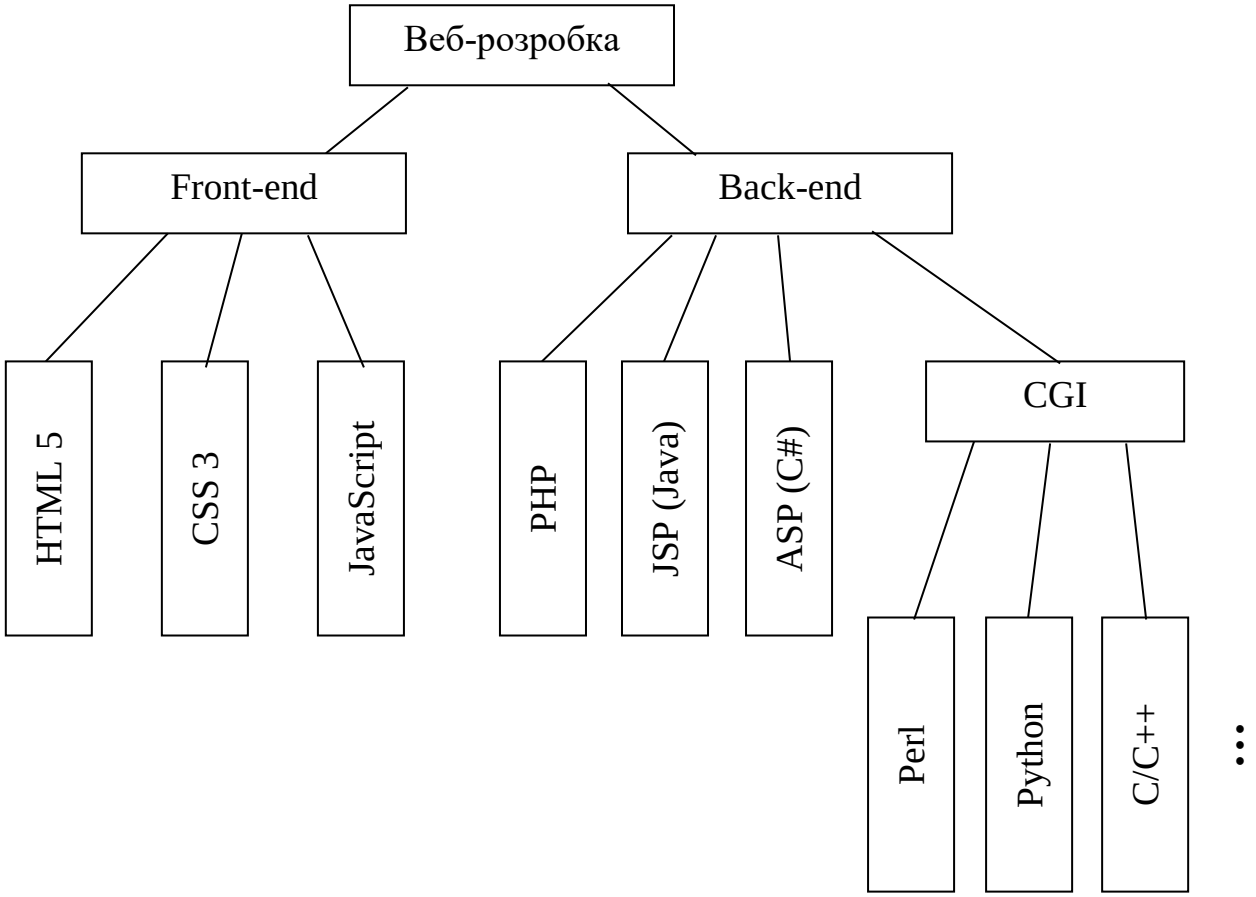

Рис. 3.4. Найпопулярніші засоби розробки сучасних Інтернет-ресурсів.

Якщо конкретніше говорити про веб-програмування, то тут ситуація є максимально поляризованою (ще більше, ніж у описаному вище випадку написання фронт-енду одним програмістом для бек-енд-утиліти, створеної іншим розробником), оскільки користувач працює і «бачить» систему через браузер на одній віддаленій машині (на комп'ютері, що називається клієнтом), а логіка додатку та практично уся супутня інформація розміщується на зовсім іншому комп'ютері-сервері.

Таким чином, усі програмні коди та висхідні тексти, що виконуються у браузері, тобто на стороні клієнта, відносяться до front-end частини. З іншого боку, все, що виконується на сервері, відноситься до back-end складової.

Якщо глибоко не вдаватися у детальний аналіз, то коротко можна сказати, що набір інструментів для front-end розробки значно менший, ніж для беку. Серед них можна виділити:

- HTML – п’ятої версії (HTML5), найсучасніший стандарт мови гіпертекстової розмітки (HyperText Markup Language), де традиційно під гіпертекстом мають на увазі текст, оснащений гіперпосиланнями на інші частини документів та, можливо, супроводжуваний картинками, звуками, відео, і т.п.;

- CSS – третьої версії (CSS3) – правила запису стилів різноманітних елементів веб-сторінки, або, якщо говорити точно – каскадні таблиці стилів;

- JavaScript – найпотужніший інструмент front-end розробки, що дозволяє вивести html-сторінки на новий рівень практично повноцінних додатків, на зразок настільних, що активно реагують на дії користувача, передають і отримують інформацію, від нього.

Ці технології є безальтернативними, тому розглянемо їх докладніше.

1) Hyper Text Markup Language (HTML) - мова розмітки гіпертексту - призначена для написання усіх документів в мережі World Wide Web - WWW.

HTML документ - це текстовий файл, який має спеціальні мітки, що називаються тегами, які при затребуванні клієнтом передаються із комп’ютера-серверу на браузер клієнта і використовуються ним для відображення вмісту файлу па екрані комп'ютера.

За допомогою цих міток можна виділяти заголовки документа, змінювати колір, розмір і написання літер, вставляти графічні зображення і таблиці. Але основною перевагою гіпертексту перед звичайним текстом є можливість додавання до вмісту документа гіперпосилань - спеціальних конструкцій мови HTML, які дозволяють клацанням миші перейти до

перегляду іншого документа. Таким чином «гіпертекст» означає «надтекст» - розвинення ідеї текстового документу, але із більш широкими можливостями.

Сама по собі мова HTML є різновидом більш загальної мови XML (eXtensible Markup Language), причому з одного боку звужує її властивості, а з іншого – навпаки деталізує. Так, у мові XML допустимим є використання будь-яких тегів, тобто з довільними іменами, в той час, як назви тегів мови HTML є цілком фіксованими, а інші слова окрім стандартних, використовувати не можна. З іншого боку, той вміст веб-сторінки, що розміщений біля конкретних тегів (точніше – між парою однакових тегів), набуває визначених стандартом властивостей (на відміну від тексту, що розміщений поруч із тегами XML, які самі по собі практично нічого не означають, а використовуються лише при наявності певних домовленостей про відображення й обробку того, чи іншого тегу) відображується браузером відповідно до суті цих тегів: якщо тегом є <B>, то текст зображуватиметься жирним, <i> - курсивом, і т.д.

HTML-документ має дві складові: власне текстова інформація, тобто дані, що складають вміст документа, і теги - спеціальні конструкції мови HTML, які використовуються для розмітки документа і керують його відображенням. Теги мови HTML визначають, в якому вигляді буде представлений текст, які його компоненти будуть виконувати роль гіпертекстових посилань, які графічні або мультимедійні об’єкти повинні бути включені в документ.

Графічна та звукова інформація, що включається в HTML-документ, зберігається в окремих файлах. Програми перегляду HTML-документів називають браузерами, і вони інтерпретують теги розмітки і оперують текстом і графікою, розміщуючи їх на екрані відповідним чином. Для файлів, що містять HTML-документи використовується розширення .htm або .html.

У переважній більшості випадків теги використовуються парами. Пара складається з тега, що відкриває <ім’я_тега> і тега, що закриває </ім’я_тега>.

Дія будь-якого парного тега починається з того місця, де зустрівся відкриваючий тег, і закінчується при зустрічі відповідного закриваючого тега. Часто пару, що складається з відкриваючого і закриваючого тегів, називають контейнером, а частину тексту між відкриваючим і закриваючим тегом - елементом.

Послідовність символів, яка утворює текст, може складатися з пробілів, табуляцій, символів переходу на новий рядок, символів повернення каретки, букв кириличного та латинського написань, знаків пунктуації, цифр, і спеціальних символів (наприклад #, +, $, @), за винятком наступних чотирьох символів, що мають в HTML спеціальний сенс: < (менше), > (більше), & (амперсанд) і «(лапки)». Якщо необхідно включити в текст будь-який із цих спецсимволів, то слід закодувати його особливою послідовністю символів:

```
<    -    &lt
>    -    &gt
&    -    &amp
“    -    &quot
```

Найпершим із тегів HTML розміщується однойменний тег <html>. Він завжди відкриває документ, так само, як тег </html> повинен неодмінно стояти в останньому його рядку. Ці теги позначають, що рядки, які між них знаходяться, представляють собою єдиний гіпертекстовий документ. Без цих тегів браузер або інша програма перегляду не в змозі ідентифікувати формат документа і правильно його інтерпретувати.

Далі, якщо говорити укрупнено, HTML-документ має дві частини: заголовок (head) і тіло (body), розташованих в наступному порядку:

```
<Html>
<Head> Заголовок документа </ head>
<Body> Тіло документа </ body>
</ Html>
```

Найчастіше в заголовок документа включають парний тег <title> ... </title>, що визначає назву документа. Багато програм перегляду використовують його як заголовок вікна, в якому виводиться документ.

Програми, що індексують документи в мережі Інтернет, використовують назву для ідентифікації сторінки. Гарна назва повинна бути досить довгою для того, щоб можна було коректно вказати відповідну сторінку, і в той же час воно має міститися в заголовку вікна. Назва документа вписується між відкриваючим і закриваючим тегами title.

Тіло документа є обов'язковим елементом, так як в ньому розташовується весь матеріал веб-сторінки. Тіло документа розміщується між тегами <body> і </body>. Все, що розміщено між цими тегами, інтерпретується браузером відповідно до правил мови HTML дозволяють коректно відображати сторінку на екрані монітора.

Текст в HTML розділяється на абзаци за допомогою тега . Він розміщується на початку кожного абзацу, і програма перегляду, зустрічаючи його, відокремлює абзаци один від одного порожнім рядком. Використання закриваючого тега є необов'язковим.

Якщо потрібно «розірвати» текст, перенісши його залишок на новий рядок, при цьому, не виділяючи нового абзацу, використовується тег розриву рядка
. Він змушує програму перегляду виводити символи, що стоять після нього з нового рядка. На відміну від тега абзацу, тег
 не додає порожній рядок. У цього тега немає парного закриваючого тега.

Мова HTML підтримує логічне н фізичне форматування вмісту документа. Логічне форматування вказує на призначення даного фрагмента тексту, а фізичне форматування задає його зовнішній вигляд.

При використанні логічного форматування тексту браузером виділяються різні частини тексту відповідно до структури документа. Щоб відобразити назву, використовується один з тегів заголовка. Заголовки в типовому документі поділяються за рівнями. Мова HTML дозволяє задати шість рівнів заголовків: h1 (заголовок першого рівня), h2, h3, h4, h5 і h6. Тема першого рівня має зазвичай більший розмір і насиченість в порівнянні з заголовком другого рівня. Приклад використання тегів заголовків:

```
<Hl> l. Назва розділу </hl>
<H2> l.l. Назва підрозділу </h2>
```

Теги фізичного форматування безпосередньо задають вид тексту на екрані браузера, наприклад пара   виділяє текст напівжирним шрифтом, <u> </u> задає підкреслення тексту,   керує шрифтом тексту. Саме ці елементи вважаються застарілими, і замість них у специфікації HTML5 рекомендується користуватися стилями, що розглянемо нижче.

Тег <img> вставляє зображення в документ, причому так, якби воно було просто одним великим символом. Закриваючий тег для зображення не потрібний. Приклад застосування тега:

```
<Img src = "picture.gif">
```

Для створення гіпертекстового посилання використовується пара тегів <a> ... . Фрагмент тексту, зображення або будь-який інший об'єкт, розташований між цими тегами, відображається у вікні браузера як гіпертекстове посилання. Активація такого об’єкта за допомогою щигля мишею призводить до завантаження у вікно браузера нового документа або до відображення іншої частини поточної Web-сторінки. Гіпертекстове посилання формується за допомогою формули:

```
<A href = "document.html"> посилання на документ </a>
```

Href тут є обов'язковим атрибутом, значення якого і є URL-адресою запитуваного ресурсу. Лапки в завданні значення атрибута href не обов'язкові (як і при завданні значень і для інших атрибутів будь-яких тегів, але використання лапок – подвійних, чи одинарних, є вкрай бажаним). Якщо задається посилання на документ на іншому сервері, то вид гіперпосилання такий:

```
<A href = "http://www.school.dn.ua/11.jpg">Фотографія 11-А
</ a>
```

Підсумовуючи можливості мови HTML, можна сказати, що за допомогою різних тегів можна малювати таблиці, форматувати текст, вставляти в документ зображення, відео-, звукові файли та інше. В процесі

свого розвитку все більша увага приділялася тегу <style> та пов’язаними з ним каскадними таблицями стилів.

Каскадні таблиці стилів (Cascading Style Sheets, CSS) - це мова, яка містить набір властивостей для визначення зовнішнього вигляду документа. Специфікація CSS (CSS3 – на даний момент) визначає властивості і описову мову для встановлення зв'язку властивостей з елементами в документі. Розуміння таблиць стилів необхідно для додавання динамічного стилю сторінки. Під динамічним стилем (dynamic style) тут маємо на увазі модифікацією таблиці стилів, пов'язану з документом за допомогою сценарію.

На вузлі консорціуму W3C (www.w3.org) можна знайти останню інформацію про нововведення і елементах, підтримуваних таблицями стилів.

Таблиці стилів представляють собою абстракцію, в якій стиль документа визначається окремо від змісту або структури. Існує три методи додавання таблиць стилів в документ, доступних для Web-майстра - в цілому, з підвищенням рівня складності розширюються надані можливості з одночасним збільшенням ступеня абстракції. Перший метод полягає в використанні таблиці внутрішніх стилів (inline style sheet). Внутрішні стилі (inline styles) визначаються безпосередньо в елементі. Другий метод полягає в використанні таблиці глобальних стилів (global style sheet) для визначення стилю на початку документа. Третій, найбільш абстрактний і потужний метод полягає в використанні таблиці пов'язаних стилів (linked style sheet) для визначення стилю окремо в іншому документі.

Внутрішні стилі мало відрізняються від традиційного HTML. При використанні внутрішніх стилів зовнішній вигляд документа важко змінити. Перевага даного методу полягає в скороченому обсязі розмітки і в тому, що HTML може бути більш повноцінно використаний для подання вмісту презентації. Використання таблиці глобальних стилів дозволяє більш ефективно відокремити представлення від вмісту, а також дає можливість швидкої і незалежної зміни стилю і обробки документа. Використання

таблиці пов'язаних стилів дає більше переваг, представляючи вміст у вигляді набору сторінок або визначаючи цілий Web-вузол за допомогою одного файлу.

Термін cascading (каскадні) в назві CSS вказує на можливість злиття різних таблиць стилів для створення єдиного визначення стилю для елемента або для цілого документа. Це дозволяє проводити передбачуване злиття таблиці стилів Web-вузла з таблицею стилів документа і навіть з внутрішнім стилем.

Отже, внутрішній стиль (inline style) є по суті таблицею стилів для одиночного екземпляру елемента і його визначено безпосередньо у тегу елемента. Таблиця внутрішніх стилів визначається з використанням атрибута STYLE, а дані для атрибута визначаються за допомогою мови таблиці стилів. Нижче наведено код HTML, який збільшує шрифт відображення вмісту параграфа і вирівнює параграф по центру на жовтому фоні:

```
     <P STYLE = "font-size: 120%; text-align: center; background: yellow">
     Створює жовтий вирівняний по центру параграф з великим розміром шрифту.
     </ P>
```

Внутрішні стилі допомагають при вивченні мови CSS або за необхідності швидко змінити одиночний екземпляр елемента. Однак, внутрішні стилі не відповідають ідеології структурованого документа і погано працюють при необхідності зміни зовнішнього вигляду ряду елементів в документі, коли презентація та вміст розділені в повному обсязі. Для відділення стилю документа від його структури таблиця стилів повинна бути визначена в заголовку документа або як окремий файл, який пов'язаний з документом.

Наступною, більш ідеологічно вірною можливістю, є використання окремого тегу <STYLE> для завдання таблиць глобальних стилів, який зазвичай розміщується в заголовку документа. Розміщення усіх стилів документа в одному місці спрощує зміну режиму відтворення документа. Наведений нижче приклад таблиці стилів визначає зовнішній вигляд усіх

параграфів в документі. Для зміни режиму відтворення параграфів потрібно змінити тільки елемент STYLE. Якби використовувалися внутрішні стилі, то довелося б міняти кожен параграф в документі окремо.

```
<HTML>
  <HEAD>
     <STYLE TYPE = "text / css">
        P {font-size: 120%; text-align: center; background:
yellow}
     </ STYLE>
  </ HEAD>
  <BODY>
     <P> Все параграфи тепер більше і вирівняні по центру
на жовтому
        тлі. </ P>
  </ BODY>
</ HTML>
```

Для зв'язку стилю з певним елементом використовується селектор (selector). У наведеному вище прикладі був створений простий селектор, який пов'язаний зі стилем у всіх параграфах. Можуть бути також визначені більш функціональні контекстуальні селектори, що описуються нижче у цьому розділі.

Ще одним із трьох способів завдання стилів є введення таблиці пов'язаних стилів (linked style sheet), яка знаходиться у зовнішньому файлі. Перевага використання таблиці пов'язаних стилів полягає в тому, що всі правила і стилі можуть бути визначені і вміщені в одному файлі, який може бути спільно використаний багатьма сторінками або навіть цілим Web-вузлом. При використанні таблиці пов'язаних стилів обробка всіх параграфів цілого Web-вузла може бути змінена в одному документі. Таблиця пов'язаних стилів може також підвищити продуктивність, оскільки вона кеширується локально на диску клієнта, окремо від документа, так що кожен документ має менший розмір, а інформацію про стилі буде потрібно завантажити тільки один раз.

Для визначення таблиці пов'язаних стилів у заголовку документа розміщується тег <LINK>:

```
<HTML>
  <HEAD>
```

```
          <LINK REL = "stylesheet" TYPE = "text / css" HREF =
"fancy.css">
        </ HEAD>
        <BODY>
          <P>  This  document  uses  the  styles  specified  in
fancy.css. </ P>
        </ BODY>
     </ HTML>
```

Атрибут REL визначає, що пов'язаний файл є таблицею стилів, а атрибут TYPE визначає тип MIME таблиці стилів. Атрибут HREF є покажчиком URL, що вказує на зовнішню таблицю стилів. Таблиця пов'язаних стилів повинна містити тільки контекстуальні правила і визначення стилю і не може включати код HTML.

Для створення таблиці стилів всередині документа використовується той же синтаксис, який використовувався при створенні таблиці пов'язаних стилів. Мова CSS складається з селекторів і правил подання (presentation rules). Селектори (selectors) визначають елементи, які пов'язані з певним правилом, а правила подання встановлюють методи обробки даних елементів.

CSS містить два типи селекторів: прості і контекстуальні. Простий селектор пов'язує елемент на основі його атрибутів або типу незалежно від контекстуального положення усередині інших елементів. Контекстуальні елементи є більш потужними - вони можуть зв'язати правило з контейнерами певного елемента, наприклад, усі теги <EM> всередині тегів <P>.

У базовій формі простий селектор може бути створений для зв'язку певного елемента, класу елементів або ідентифікатора (ID) з певним стилем. Наведений нижче код демонструє ряд простих селекторів і їх правил подання:

```
      <STYLE TYPE = "text / css">
        / * Change all H1s to red. * /
        H1 {color: red}
        / * Встановлює напівжирний шрифт всіх елементів з тегом
CLASS = "Special" boldface. * /
        .special {font-weight: bold}
        / * Розміщує елемент з ідентифікатором ID = special на
жовтому тлі.*/
        #special {background: yellow}
```

```
        / * Встановлює висновок елементів H1 з тегом CLASS =
"cool" з великим інтервалом.* /
        H1.cool {letter-spacing: 2px}
    </ STYLE>
```

Селектори можуть бути перераховані через кому, що дозволяє одночасно описати кілька селекторів:

```
    / * Встановлює для всіх заголовків однакові правила. * /
    H1, H2, H3, H4, H5, H6 {color: red; background: yellow}
```

Контекстуальні селектори визначають ієрархію контейнерів, з якою повинен бути пов'язаний стиль. Ієрархія контейнерів визначається порядком елементів в списку через кому. Наприклад, наведений нижче оператор визначає правило для всіх елементів EM, що знаходятцься в елементі P:

```
    P EM {color: blue}
```

Кожен селектор може посилатися на теги CLASS, ID або тип елемента. Нижче наведена більш складна версія контекстуального селектора:

```
     / * Будь-який елемент з CLASS = "cool", який знаходиться
всередині елемента LI з CLASS = "special" і далі перебуває
всередині елемента UL, буде використовувати даний стиль. * /
    UL LI.special .cool {font-weight: bolder; font-size: 120%}
```

Всі елементи контекстуального селектора є нечутливими до регістру - наприклад, .cool це те ж саме, що і .cOoL.

Псевдоклас (pseudo-class) складається з елементів одного типу, які задовольняють певному контекстуальному критерію. Наприклад, переглянуті елементи Anchor (посилання) представляють собою псевдоклас visited. Активні і не переглянуті посилання являють собою псевдокласи active і link, відповідно. Псевдоклас в таблиці стилів відділяється двокрапкою:

```
    A: link {color: green}
    : Link {color: green}
```

У другому прикладі опущено ім'я елемента (A), так як тільки посилання мають псевдоклас link. Псевдоклас може бути використаний так само, як клас або покажчик ID і є нечутливим до регістру.

На одні й ті ж самі елементи можуть посилатися кілька селекторів. CSS визначає послідовність каскадування (cascading order), яка використовується

для вирішення проблем перекривання областей дії селекторів і правил. Послідовність каскадування об'єднує всі правила, які застосовуються до елементу, шляхом сортування на основі їх визначень. Наприклад, елемент Strong, що знаходиться в елементі H1, може мати правила подання, визначені селектором H1, селектором STRONG і контекстуальним селектором для елементів Strong всередині елементів H1. Параметр каскадування в CSS визначає порядок об'єднання даних трьох правил. Загалом, правило для більш конкретного контекстуального селектора відкидає правило для менш конкретного селектора, а правила, подані нижче у початковій таблиці стилів або документі, мають більш високий пріоритет.

Докладно розглянувши каскадні таблиці стилів слід також подивитися основні особливості останнього, дуже потужного засобу front-end-розробки, а саме – мови програмування JavaScript. Це спеціалізована мова програмування, яка традиційно використовується для управління об’єктною моделлю документа у браузері під час перегляду сторінок мережі Інтернет (існують продукти, що перетворюють JavaScript на мову загального призначення, команди якої виконуються на сервері, наприклад, Node.JS; але такий підхід сильно протирічить початковій концепції використання цієї мови програмування, і, за умови наявності значної кількості традиційних засобів back-end-розробки перетворення на серверну мову ще й JavaScript значна кількість програмістів вважає абсолютно недоцільною, тому про цей варіант використання говорити більше не будемо).

Особливістю JavaScript є те, що його висхідні програмні коди інтерпретуються, що є досить гнучким, але повільним рішенням.

Оператори у цій, в цілому C-подібній мові, розділяються крапкою з комою. Мова чутлива до регістру, що часто випускають з виду початківці, ймовірно тому, що HTML, застосовуваний зазвичай з JavaScript спільно, не залежить від регістру (імена тегів і атрибутів HTML можна писати як малими, так і великими літерами).

Однорядковий коментар виділяється символом //, а багаторядковий - парою символів / * і * /, в чому JavaScript знову повторює С.

До даних застосовується слабкий (динамічний) контроль типів. В операторах з різнотипними даними останні автоматично приводяться до необхідного типу. Типи даних можуть бути примітивними і складеними. Примітивні типи містять прості однорідні значення, такі дані можна передавати функціям як параметри за значенням, а не за посиланням. Складені типи містять різнорідні дані (в тому числі і складені), їх передають у функції тільки за посиланням.

Мова JavaScript об’єктно-орієнтована, проте заснована на прототипах, а не на класах. Є чотири типи об’єктів: вбудовані об’єкти, об’єкти браузера, об’єкти документа і об’єкти користувача (програміста).

Введення-виведення в основному обмежене взаємодією з документами і користувачами. За умовчанням передбачається, що доступ до локальної файлової системи заборонений. Однак браузери можуть надавати спеціальні об’єкти, за допомогою яких забезпечується робота з файловою системою користувача, хоча і з видачею попереджень про небезпеку виконання файлових операцій.

Сценарії JavaScript активно взаємодіють з об’єктами, вбудованими в Web-сторінку. Для цього вони, власне, і створюються. Але перш, ніж ця взаємодія стане можливою, слід впровадити код сценарію в текст HTML-документа. Існує кілька способів зв'язати HTML-документ з конкретним сценарієм (скриптом), але зазвичай їх просто розміщують всередині контейнерного тега <SCRIPT>, тобто між дескрипторами <script> і </script>.

Контейнер <SCRIPT> в цьому випадку буде перебувати безпосередньо в HTML-документі, причому у довільному його місці. Програмний код пишуть прямо в HTML-документі або в спеціальних текстових файлах, які можна викликати з головного HTML-документа. Для початку розглянемо перший варіант. Перш за все браузер знаходить тег <script> в тілі веб-документа, і весь наступний текст намагається обробити як скриптовий код. І

так до тих пір, поки не зустріне закриваючий тег </script>. Після цього всі наступні символи будуть вважатися HTML-текстом. Будь-який HTML-документ може містити довільне число «скриптових включень», але кожне має відкриватися і завершуватися відповідним тегом. Від їх розташування у тілі HTML-документа іноді може залежати функціонування всієї Web-сторінки, але про це буде сказано пізніше.

Контейнерний тег <script> може містити атрибут SRC, який вказує ім'я або URL-адресу текстового файлу, що містить код сценарію. Цей атрибут необхідний в тому випадку, якщо сценарій розташований не безпосередньо в HTML-документі, а в окремому файлі. Розширення файлу зі сценарієм може бути яким завгодно, але зазвичай використовують js:

```
<SCRIPT SRC = «myscripts.js»> </ SCRIPT>.
```

Якщо сценарій розташовується в окремому файлі, то в ньому, зрозуміло, теги <SCRIPT> і </ SCRIPT> не пишуть. Сценарій, завантажений з зовнішнього файлу, можна уявити собі просто як його вставку в HTML-документ.

У браузерах, що потенційно підтримують сценарії, цю функцію користувач може відключити (зокрема, із міркувань підвищеної безпеки). Крім того, існують браузери, які принципово не підтримують сценарії. У даній ситуації бажано хоча б вивести повідомлення про те, що на сторінці був сценарій, але в даному конкретному випадку він не виконується. З цією метою в HTML-документі використовують контейнерний тег <noscript>, в якому розміщують текст, який з'явиться користувачеві за умови, що сценарій з тієї, чи іншої причини не виконуватиметься. Всі браузери, які підтримують сценарії, проігнорують вміст тегів <noscript> крім тих випадків, коли підтримка сценаріїв відключена. Браузери, що принципово не підтримують сценарії, навпаки, вміст тега <script> опустять (особливо, якщо він вкладений у теги <!-- -->, які є HTML-коментарем), а тега <noscript> - відобразять. Приклад наведено нижче:

```
<HTML> <HEAD>
<TITLE> Приклад застосування тега NOSCRIPT </ TITLE>
```

```
</ HEAD>
<SCRIPT TYPE = "text / JavaScript»>
<! -
alert ( «Підтримка JavaScript включена»); // ->
</ SCRIPT>
<NOSCRIPT>
<H2> Ваш браузер не підтримує JavaScript або його підтримка відключена </ H2>
</ NOSCRIPT>
</ HTML>
```

Тут був використана функція alert (повідомлення) для виведення повідомлень в маленькому діалоговому вікні.

Як уже зазначалося, в окремих файлах зазвичай розміщують бібліотеки функцій (визначення функцій), а також сценарії, які використовуються в декількох HTML-документах одного або декількох сайтів. Сценарій можна також писати у вигляді рядка операторів, між якими ставиться крапка з комою. Такий рядок, взятий в лапки, служить у якості значення атрибута-події, наприклад:

```
<IMG SRC = "mypicture.gif" ONCLICK = "alert ('Привіт!')".
```

Виклики функцій і їх визначення можуть слідувати в довільному порядку, але тільки якщо вони розташовані в одному і тому ж контейнері <script>. Якщо вони розміщуються у різних контейнерах <script>, то необхідно, щоб визначення функції передувало її виклику. Аналогічно, якщо визначення функцій знаходяться в окремому файлі, то його необхідно завантажити в HTML-документ раніше викликів цих функцій.

При спробі завантажити і виконати неправильний варіант HTML-коду з'явиться діалогове вікно з повідомленням «Помилка: Передбачається наявність об'єкта». Браузер інтерпретує теги HTML послідовно. Так, зустрівши вираз виклику функції myfunc() в одному контейнері <script>, браузер ще не має в пам'яті визначення цього об'єкта (функції), розташованого в іншому контейнері <script>. Якщо ж визначення функції і її виклик знаходяться в одному і тому ж контейнері <script>, то його вміст спочатку завантажується в пам'ять, а потім аналізується. Коли інтерпретатор

зустрічає виклик функції, то він шукає її визначення в пам'яті і в разі успіху виконує код цієї функції.

Якщо необхідно, щоб сценарій виявився в браузері перш, ніж буде завантажено елементи HTML-документа, то його слід розташувати у верхній частині HTML-коду, а ще краще в контейнері <head> в заголовку документа.

Засоби Back-end розробки.

Як зазначалося вище, під Back-end'ом мають на увазі програмні компоненти, що працюють на сервері, і вибір засобів для реалізації цих компонентів є вкрай широким – рис. 3.4, справа.

Однак, для цілей даного дослідження не потрібне підключення до бази даних, відкриття файлів на сервері і т.п., адже усю необхідну інформацію для роботи програми, вона отримує безпосередньо від користувача із форми веб-сторінки. Оскільки вся необхідна інформація для розрахунків вже є на стороні користувача, звертатися знову до серверу немає сенсу і доцільніше обмежитися обробкою даних на стороні клієнта, тобто засобами мови JavaScript.

Таким чином, у якості мов програмування для обраного напрямку (веб-розробки) обираємо HTML та JavaScript.

3.3. Вибір середовища розробки

Останнім питанням є вибір конкретного середовища розробки, що дозволяє писати код на цих обраних мовах. У якості такого середовища можна взяти популярний на сьогоднішній день серед студентської спільноти Visual Studio Code, або більш професійний PHP Storm, або простий текстовий редактор типу Sublime Editor. Така широта вибору засобів обумовлена простотою задачі (зокрема, відсутністю необхідності застосування системи управління базами даних, і т.п.) та стандартністю (поширеністю) обраних мов програмування серед представників сучасної програмістської спільноти.

У якості середовища розробки оберемо популярне завдяки своїй простоті та зручності (але достатньо функціональне) середовище Visual Studio Code, що має цікаві можливості.

Це редактор вихідних кодів, розроблений Microsoft для Windows, Linux і macOS. Позиціонується як «легкий» редактор коду для кросплатформенної розробки веб і хмарних додатків. Включає в себе відладчик, інструменти для роботи з Git, підсвічування синтаксису, IntelliSense і засоби для рефакторингу. Має широкі можливості для кастомізації: призначені для користувача теми, поєднання клавіш і файли конфігурації. Розповсюджується безкоштовно, розробляється як програмне забезпечення з відкритим вихідним кодом, але готові збірки розповсюджуються під пропрієтарною ліцензією.

Visual Studio Code заснований на Electron - фреймворку, що дозволяє з використанням Node.js розробляти настільні додатки, які працюють на движку Blink. Незважаючи на те, що редактор заснований на Electron, він не використовує редактор Atom. Замість нього реалізується веб-редактор Monaco, розроблений для Visual Studio Online.

Однією із найзручніших функцій цього середовища є згадана вище IntelliSense. Це технологія автодоповнення Microsoft, найбільш відома в Microsoft Visual Studio, яка дописує назву функції та інші ключові слова при введенні початкових літер. Крім прямого призначення, IntelliSense використовується для доступу до документації та для усунення неоднозначності в іменах змінних, функцій і методів, використовуючи рефлексію.

Як і інші системи автодоповнення, IntelliSense є зручним способом переглянути описи функцій, в тому числі переліки їхніх аргументів. Вона прискорює розробку ПО, зменшуючи кількість імен і параметрів, які програміст повинен тримати в пам'яті. Крім того, вона зменшує кількість необхідних запитів до документації, виводячи частину документації у вигляді спливаючих вікон в редакторі коду. В ході роботи IntelliSense формує в

пам'яті базу даних, що містить метадані класів, змінних і інших конструкцій, які використовуються в додатку, що розробляється. «Класична» реалізація IntelliSense працює, знаходячи в коді спеціальні маркери, такі як символ точки. Як тільки користувач вводить один з таких маркерів після імені сутності, яка містить один або кілька доступних членів (таких як змінні або методи), IntelliSense показує користувачеві спливаюче вікно зі списком відповідних членів.

Середовище Visual Studio Code має ще багато дуже зручних можливостей, через які воно, власне, і набуло своєї популярності. Це середовище взято у якості основного засобу розробки у даній роботі.

3.4. Проектування інтерфейсу користувача для програмного забезпечення, що реалізує прогнозування

Беручи до уваги специфіку проектованого програмного продукту, можна зробити висновок, що у нього буде 2 основних екрани: для збору інформації та для відображення результатів прогнозування.

Ця теза напряму витікає із алгоритму роботи користувача із програмним забезпеченням, що розробляється – рис. 3.5.

На першому екрані (рис. 3.6) при старті розміщено мінімальну кількість елементів управління для введення необхідної інформації (оскільки за умовчанням $p = 1$ – відображується в окремому полі для редагування). У кожному горизонтальному ряді розміщена інформація, що стосується r-того показника:

- перемикач для урахування чи ігнорування даного конкретного показника у кінцевій формулі;
- ваговий коефіцієнт показника;
- поточне значення показника для першої команди-суперника;
- поточне значення показника для другої команди;
- значення показника для першої команди-суперника в минулому матчі;
- значення показника для другої команди-суперника в минулому матчі.

Рис. 3.5. Алгоритм роботи проектованого ПЗ.

Якщо користувач системи збільшує число *p*, то у кожному горизонтальному ряді справа приписується по два поля для введення (для команди 1 та для команди 2). Динамічна зміна інтерфейсу реалізована мовою JavaScript.

Рис. 3.6. Зовнішній вигляд екрану для введення інформації.

Після введення усієї необхідної інформації користувач повинен натиснути кнопку «Розрахувати» і програма переходить до екрану виведення результату – рис. 3.7.

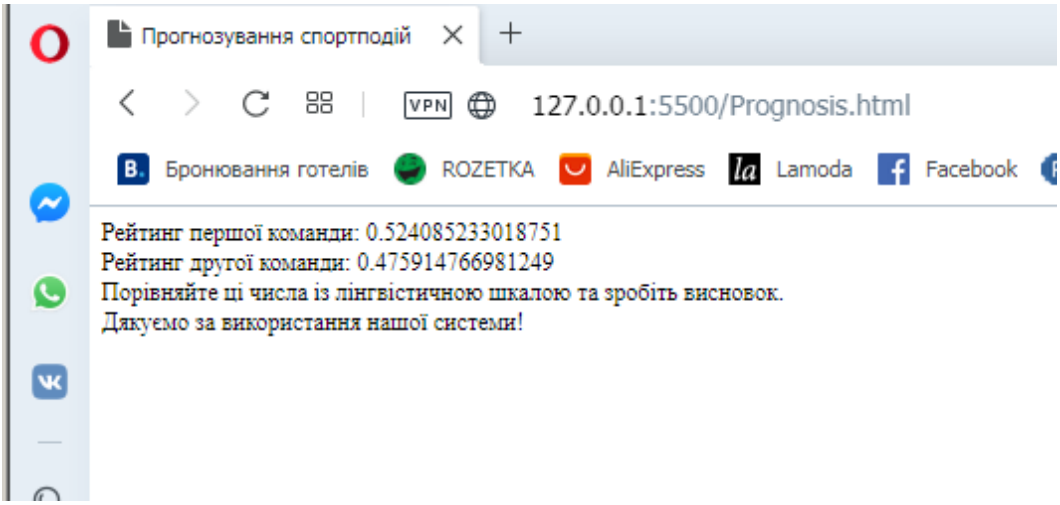

Рис. 3.7. Екран програмного забезпечення для виведення інформації.

В процесі введення користувач може допускати помилки, зокрема введення нечислових даних. Система це автоматично відслідковує і очищує поле, яке не може перетворити на число – рис. 3.8.

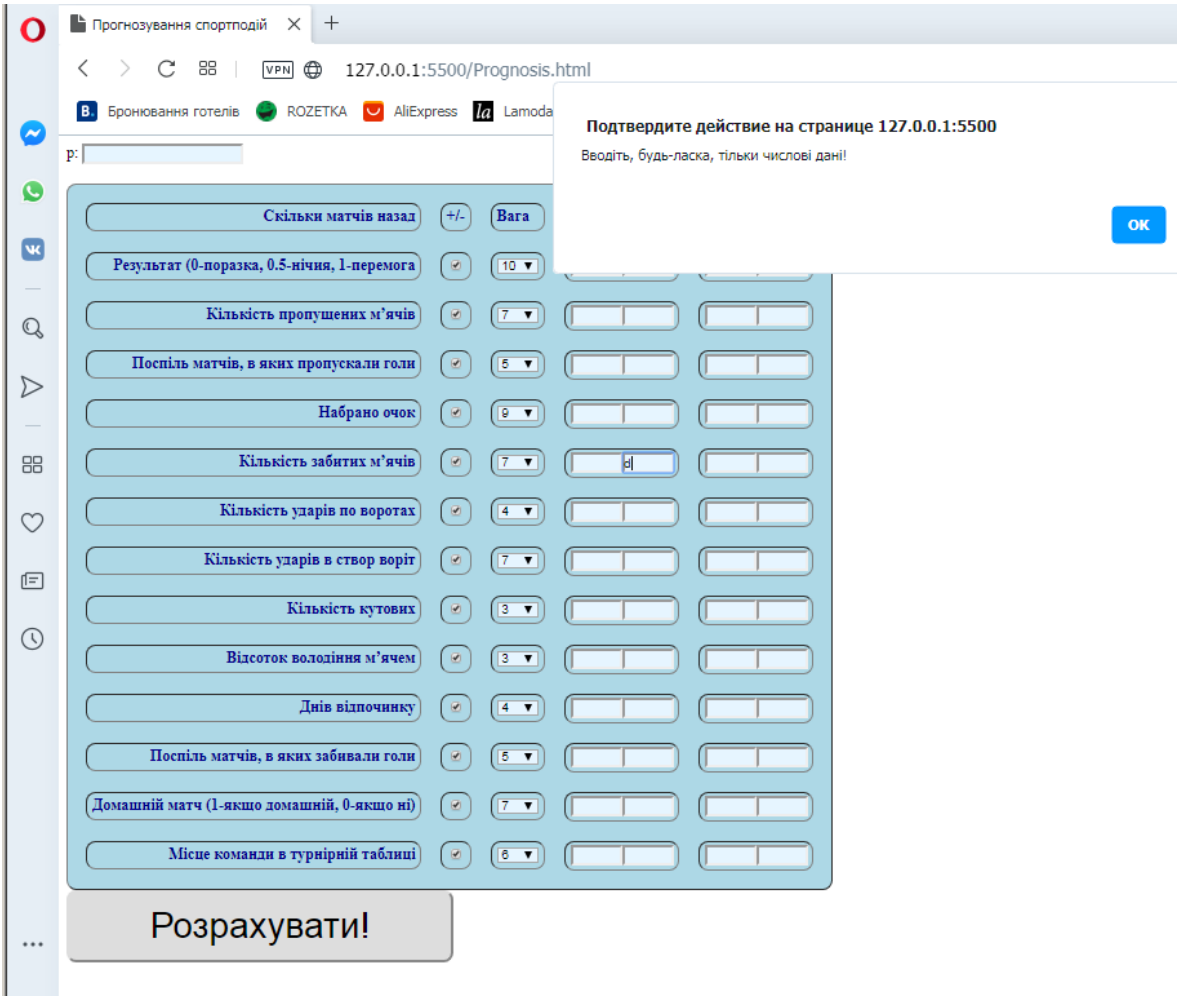

Рис. 3.8. Приклад повідомлення про введення некоректних даних.

Висновки по розділу 3.

Таким чином, у розділі розглянуто особливості проектування системи для прогнозування результатів спортивних подій (конкретне програмне рішення виконується для прикладу футболу).

Розглянуто сучасні технології програмування (об’єктно-орієнтовану та структурну) та обрано структурний підхід, як більш простий, що відповідає особливостям задачі (невелика с функціональної точки зору, але складна математично). Обрано середовище розробки, у якості якого рекомендується використовувати Visual Studio Code. Програмний продукт запропоновано створити у вигляді веб-додатку для підвищення його універсальності, кросплатформенності, організації можливості доступу до нього з будь-якого ПК, підключеного до Інтернет із довільної точки світу.

Також спроектовано графічний інтерфейс користувача, який є достатньо простим, зважаючи на однорідність та структурованість вхідної інформації. Фактично, створена програма працює в режимі, близькому до пакетного: на першому екрані інформація збирається, потім ініціюється розрахунок і здійснюється перехід до наступного екрану виведення інформації.

Таким чином, можна переходити до програмної реалізації розробленого рішення, що базується на удосконаленій методиці прогнозування результатів спортивних подій.

РОЗДІЛ 4. РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТІВ ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1. Особливості програмної реалізації алгоритмів прогнозування результатів спортивних подій

Програмна реалізація виконувалася мовою HTML, а різноманітні перевірки, дії по збору та перевірці вхідної інформації, динамічному відображенні результатів виконуються мовою JavaScript.

Код розробленого програмного забезпечення наведено у додатках 1-3.

Оскільки у попередньому розділі у якості технології програмування було обрано структурний підхід, то весь текст програми мовою JavaScript розбитий на функції:

- bindhandlers() – зв’язує обробники подій клацання на кнопку «Розрахувати» та зміни вмісту текстового поля для введення числа p із відповідними подіями. Також перший раз викликає функцію filldiv() для заповнення таблиці показників двома колонками (за умовчанням);

- checknum() – перевіряє значення, введене у будь-яку комірку таблиці на числову форму: якщо із вводу користувача не вдається створити число, то вміст комірки пропадає, а користувачу видається попереджувальне повідомлення;

- filldiv() – викликається кожного разу, коли необхідно перебудувати таблицю з полями для введення показників. Це, наприклад, відбувається, коли користувач вводить інше значення p;

- startcalc() – безпосередньо реалізує математичну функцію (2.11), зчитуючи вхідні дані із таблиці. Функція виконана стійкою, і не викликає помилок навіть за умови відсутності певних необхідних даних (відповідний рядок тоді просто ігнорується).

Вказаної структури цілком вистачає для організації повної функціональності спроектованого програмного забезпечення (в першу чергу,

зважаючи на його пакетний характер роботи: «введення вхідних даних»-«розрахунок»-«видача результату»).

4.2. Документаційне забезпечення розробленого програмного забезпечення

До створеного програмного забезпечення розроблено інструкцію користувача, що складається з наступних пунктів:

1) Перед початком роботи з програмним продуктом слід впевнитися, що в наявності присутні усі три його компоненти: Prognosis.html, mycss.css, myscript.js.

2) Для початку роботи з програмою завантажте у браузері файл Prognosis.html.

3) Оберіть ті показники, які будуть використовуватися для прогнозування і впевніться, що біля їхніх назв установлені перемикачі (галочки), а біля назв тих показників, що не плануються до використання, перемикачі мають бути знятими.

4) Перевірте значення вагових коефіцієнтів для тих показників, що будуть використовуватися для прогнозування; за необхідності змініть значення за умовчанням.

5) Виберіть кількість *p* попередніх матчів, що будуть ураховуватися (в самій верхній лівій частині вікна).

6) Заповнити усі необхідні текстові поля відомими числами. У ліве поле кожної клітинки таблиці заноситься показник першої команди, а у праве поле – показник другої команди.

7) Особливу увагу слід звернути на показники «Результат» та «Домашній матч», які заповнюються вручну на основі відомих про команду даних:

   a. якщо якийсь матч для команди є домашнім, то у відповідну комірку слід поставити 1, інакше – 0;

b. якщо якийсь матч команда виграла (із попередніх) то у комірку слід поставити 1, при нічиїй – 0.5, при поразці – 0.

8) Після заповнення усіх комірок, що відмічені галочками, натиснути на кнопку «Розрахувати».

9) Дочекатися результату і звірити число першої команди з табл. 2.1, отримавши її результат. Результат другої команди буде дзеркально симетричним (якщо перша виграла, друга програє; якщо перша дає нічию, та ж ситуація буде і з другою, і т.д.).

10) Для завершення роботи з програмою слід закрити вікно браузера.

4.3. Тестування розробленого програмного забезпечення та аналіз результатів його роботи

В першу чергу, створене програмне забезпечення тестувалося на стабільність роботи та адекватність роботи у крайніх випадках.

Встановлено, що система працює надійно, без помилок. Присутній захист від введення некоректних даних працює добре.

При введенні абсолютно рівних показників система показала 50% на 50%.

При введенні показників одної команди практично усюди нульових (або дуже великих – для дестимуляторів), а іншої – звичайних, то система показала майже 100% перемогу другої команди.

Розроблене програмне забезпечення було також протестоване на матчах Української вищої ліги і результати, показані створеною програмою дали біля 70% співпадінь, на відміну від базового рішення [1], що надає точність прогнозування біля 60%. Таким чином, за рахунок впровадження сумування інформації за часом, вдалося підвищити якість прогнозування приблизно на 10%.

Висновки по розділу 4

Таким чином, у розділі розглянуто створення програмної реалізації системи прогнозування результатів спортивних подій (на прикладі футболу).

Розроблене документаційне забезпечення розробленого програмного продукту, а саме докладна «Інструкція користувача».

Виконано тестування програмного продукту, що показало його безперервну стабільну роботу, а також адекватні результати у крайніх ситуаціях (повна рівновага та тотальна перевага однієї команди над іншою).

Також виконано тестування у реальних умовах для кількох матчів Української футбольної ліги, що показало перевищення показника точності прогнозування розробленим рішенням над прототипом на величину порядку 10%.

ВИСНОВКИ

В роботі досліджено актуальну проблему ефективного прогнозування результатів проведення спортивних подій. Проаналізовано існуючі методики прогнозування і встановлено, що вони або базуються лише на параметрах того заходу, результат якого прогнозується, або є продовженням статистично дослідженої послідовності результатів учасника змагань (шляхом побудови часового тренду). У якості основних методів розрахунків розглядається апарат нейронних мереж та засоби нечіткої логіки, але за основу для подальшого удосконалення узято більш надійний (передбачуваний) з математичної точки зору кваліметричний метод.

На основі проведеного аналізу існуючих методів прогнозування результатів спортивних подій запропоновано удосконалення, що враховує два типи вхідної інформації: параметри майбутнього заходу, що прогнозується, а також результати попередніх подій, пов’язаних із учасниками майбутнього заходу. Фактично запропоновано певний синтез статистичних методів та кваліметричних (скорингових) моделей.

Після опрацювання математичного підґрунтя задачі, розглянуто засоби та технології, за допомогою яких можна реалізувати дані розрахунки. Обрано структурний підхід до програмування, як більш простий, та такий що у повній мірі відповідає алгоритмічно простій задачі («збір вхідних даних»-«математичний розрахунок»-«виведення результату»). Програмний продукт вирішено зробити у вигляді веб-додатку із міркувань більшої універсальності, кросплатформенності. Мовами розробки обрано стек HTML+JavaScript, в першу чергу через їх широку розповсюдженість для вирішення задач фронт-енд розробки.

В роботі розроблено інтерфейс користувача системи, виконано програмну реалізацію, яку протестовано для кількох матчів, в результаті чого отримано задовільні результати прогнозування. Для створеного програмного забезпечення розроблено необхідне документаційне забезпечення.

Програмний продукт може застосовуватися на практиці для прогнозування результатів реальних спортивних подій, а саме футбольних матчів. В перспективі для покращення точності розрахунків можливе підключення елементів штучного інтелекту, зокрема нечіткої логіки, розширення кількості показників, що ураховуються, прийняття до уваги все більшої кількості подій, що передували прогнозованій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Бєлка Д.О., Аверкина М.Ф. Моделі прогнозування футбольних матчів [Електронний ресурс]. URL: https://naub.oa.edu.ua/2019/%d0%bc%d0%be%d0%b4%d0%b5%d0%bb%d1%96-%d0%bf%d1%80%d0%be%d0%b3%d0%bd%d0%be%d0%b7%d1%83%d0%b2%d0%b0%d0%bd%d0%bd%d1%8f-%d1%84%d1%83%d1%82%d0%b1%d0%be%d0%bb%d1%8c%d0%bd%d0%b8%d1%85-%d0%bc%d0%b0%d1%82/
2. Дідківська В.А. Прогнозування результатів спортивних змагань за допомогою марківських моделей та нейронних мереж: дис. магістра. К.: НТУУ «КПІ», 2018. 123 с.
3. Кулик В. М, Коротєєва Т. О. Алгоритм прогнозування результатів футбольних матчів на основі нейронних мереж. Науковий вісник НЛТУ України, 2017. Т. 27. № 9. С.111-114.
4. Полухін О.А., Шаров С.В. Використання нейромережі для прогнозування результатів футбольних матчів. Інформаційні технології в освіті та науці: зб. наук. пр. 2018. №10. С.214-217.
5. Семенюк В. О. Математичні моделі прогнозування результатів футбольних матчі. XII Міжн. наук.-практ. конф. «Інформаційні технології і автоматизація – 2019», Одеса, 17-18 жовтня 2019 : зб. доп. Одеса, 2019. Ч. 1. С. 10-12.
6. Терентьєв Р.А. Прогнозування потреби в ресурсах для серверної системи в умовах хмарних обчислень: дис. магістра. К.: НТУУ «КПІ», 2018. 97 с.
7. Штоба С. Д., Цаконас А. Д,. Дуніас Г. Д. Прогнозування результатів футбольних матчів за допомогою машини опорних векторів. Вісник Житомирського інженерно-технологічного інститута. 2003. С. 181-186.
8. Behter L.V., Klevets N.I. Weighted sum of indexes method for predicting football matches [Електронний ресурс]. Best.Today. URL: http://bets.today/ru/articles/weighted-sum-of-indexes.

9. Douwe B. Predicting sports events from past results. University of Twente. 2011. C. 1-6.
10. Igiri, Chinwe P. An improved prediction system for Football match result / P. Igiri, Chinwe, O. Nwachukwu, Enoch. Department of Computer Science, University of Port Harcourt. 2014. C. 12-20.
11.. Somboonphokkaphan A. Tennis Winner Prediction based on Time-Series History with Neural Modeling // IMECS 2009: International Multi-Conference of Engineers and Computer Scientists, Vol. I and II, I, 2009. P.127-132.
12. Albin Y. Predicting outcomes of Soccer matches using machine learning / Y. Albina. Saint-Petersburg State University Mathematics and Mechanics Faculty. 2014. C. 3-12.
13. Kushal G. Football Match Winner Prediction / G. Kushal, S. Harshal, V. Saurabh, D. Khushali. Department of Computer Engineering, Dwarkadas J. Sanghvi College of Engineering, Mumbai, India. International Journal of Emerging Technology and Advanced Engineering, volume 5, issue 10, October 2015. C. 364-368.
14. Ben U. Predicting Soccer Match results in the English Premier League / U. Ben, F. Matthew. School of Computer Science Stanford University. 2014. C. 1-5.
15. Jongho S., Robert G. A novel way of Soccer match prediction. 2014. C. 1-5.
16. Benjamin C. Poisson Distribution: Predict the score in soccer betting. PINNACLE. 2017. 27 April. URL: https://www.pinnacle.com/en/betting-articles/Soccer/how-tocalculate-poisson-distribution/MD62MLXUMKMXZ6A8.

Додаток 1. Код веб-сторінки для прогнозування результатів спортивних подій мовою HTML.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Прогнозування спортподій</title>
    <script src="myscript.js">
    </script>
    <link rel="stylesheet" href="mycss.css">
</head>
<body>
    <div id="maindiv">
    <form>
        p: <input type="text" id="nump"><br><br>
        <div id="paramlist">
        </div>
        <input type="button" value="Розрахувати!" id="docalc">
    </form>
    </div>
</body>
</html>
```

Додаток 2. Код каскадної таблиці стилів, застосованої для розробленої веб-сторінки на CSS.

```
table, table td
{
    color:darkblue;
    background-color: lightblue;
    border-radius: 10px;
    font-weight: bold;
}

input[type="text"], select
{
    background-color: rgb(230, 245, 255);
}

input[type="button"]
{
    width:400px;
    height:75px;
    border-radius: 10px;
    font-size: 28pt;

}
```

Додаток 3. Код програми (скрипта) мовою JavaScript, застосованої для розробленої веб-сторінки.

```
const params=new Array("Результат (0-поразка, 0.5-нічия, 1-
перемога","Кількість пропущених м’ячів", "Поспіль матчів, в яких
 пропускали голи",
"Набрано очок", "Кількість забитих м’ячів", "Кількість ударів по
 воротах",
"Кількість ударів в створ воріт", "Кількість кутових", "Відсоток
 володіння м’ячем", "Днів відпочинку",
"Поспіль матчів, в яких забивали голи", "Домашній матч (1-
якщо домашній, 0-
якщо ні)", "Місце команди в турнірній таблиці");

const startw=new Array(10,7,5,9,7,4,7,3,3,4,5,7,6);

let nump=1;

function filldiv()
{
    paramlist=document.getElementById("paramlist");
    paramlist.innerHTML="";
    let str="<table border=1 cellpadding=5 cellspacing=20><tr><t
d align=\"right\">Скільки матчів назад</td><td>+/-
</td><td>Вага</td>";
    for(let i=0;i<=nump;i++)
    {
        str+="<td align=\"center\">";
        str+=""+i;
        str+="</td>";
    }
    str+="</tr>"
    for(let j=0;j<params.length;j++)
    {
        str+="<tr><td align=\"right\">"+params[j]+"</td><td><inp
ut type=\"checkbox\" id=\"chk"+j+"\" checked></td>";
        str+="<td><select id=\"sel"+j+"\">";
        for(let i=0;i<10;i++)
        {
            str+="<option";
            if(startw[j]==(i+1))
                str+=" selected";

            str+=">"+(i+1)+"</option>";
        }
        str+="</select></td>";

        for(let i=0;i<=nump;i++)
        {
            str+="<td>";
            str+="<input type=\"text\" id=\"text"+j+i+"1\" size=
3>";
```

```
            str+="<input type=\"text\" id=\"text"+j+i+"2\" size=
3>";
            str+="</td>";
        }
        str+="</tr>";
    }
    str+="</table>";
    paramlist.innerHTML=str;
    for(let j=0;j<params.length;j++)
    {
        for(let i=0;i<nump;i++)
        {
            document.getElementById("text"+j+i+"1").addEventList
ener('change',checknum,false);
            document.getElementById("text"+j+i+"2").addEventList
ener('change',checknum,false);
        }
    }
    document.getElementById("text001").disabled=true;
    document.getElementById("text002").disabled=true;
    document.getElementById("text001").style.backgroundColor="li
ghtgray";
    document.getElementById("text002").style.backgroundColor="li
ghtgray";
}

function startcalc()
{
    let maindiv=document.getElementById("maindiv");
    let res1=0,res2=0;
    let sum1,sum2,uir1,uir2,wir;
    for(let i=0;i<=nump;i++)
    {
        sum1=0;
        sum2=0;
        for(let j=0;j<params.length;j++)
        {
            if((i==0)&&(j==0))
                continue;
            if(document.getElementById("chk"+j).checked)
            {
                uir1=+document.getElementById("text"+j+i+"1").va
lue;
                uir2=+document.getElementById("text"+j+i+"2").va
lue;
                wir=+document.getElementById("sel"+j).selectedIn
dex+1;
             if(uir1+uir2==0)
                    continue;
             if((j==1)||(j==2))
                {
                    sum1+=uir2*wir/(uir1+uir2);
                    sum2+=uir1*wir/(uir1+uir2);
```

```
                }
                else
                {
                    sum1+=uir1*wir/(uir1+uir2);
                    sum2+=uir2*wir/(uir1+uir2);
                }
            }
        }
        res1+=sum1*(nump-i+1)/(nump+1);
        res2+=sum2*(nump-i+1)/(nump+1);
    }
    maindiv.innerHTML="Рейтинг першої команди: "+res1/(res1+res2
)+"<br>Рейтинг другої команди: "+res2/(res1+res2)+"<br>Порівняйт
е ці числа із лінгвістичною шкалою та зробіть висновок.<br>Дякує
мо за використання нашої системи!";
}

function checknum()
{
    const temp=+this.value;
    if(!temp)
    {
        this.value="";
        alert("Вводіть, будь-ласка, тільки числові дані!");
    }
    else
    {
        if(this.id=="nump")
        {
            nump=temp;
            filldiv();
        }
    }
}

function bindhandlers()
{
    document.getElementById("docalc").addEventListener('click',s
tartcalc,false);
    document.getElementById("nump").addEventListener('change',ch
ecknum,false);
    filldiv();
}

window.addEventListener('load',bindhandlers,false);
```